Support and Rationale Document

for the

Software Communications Architecture Specification (v2.2)

MSRC-5000SRD

V2.2

December 19, 2001

Prepared for the

Joint Tactical Radio System (JTRS) Joint Program Office

Prepared by the

Modular Software-Programmable Radio Consortium

under Contract No. DAAB15-00-3-0001

Revision Summary

Ver. No.	**Comment**
1.0	Initial release in support of initial validation of the SCAS version 1.0
2.2	Release in support of SCAS version 2.2. Only the main portion of the document was changed for this release.

Table of Contents

List of Illustrations

List of Tables

1 INTRODUCTION.

The Software Communications Architecture (SCA) Support and Rationale Document (SRD) is a companion to the Software Communications Architecture Specification (SCAS) and provides background, tutorial, and support information for the development of Joint Tactical Radio System (JTRS) software configurable radios. For easier tracking of requirements and rationale, the SRD outline is similar to the SCAS in Sections 3 through 5. Together, the SCAS and SRD provide the framework, the rationale for that framework, and examples to illustrate the implementation of the architecture for differing domains/platforms and selected waveforms. Neither document specifies implementation details for any particular domain or waveform application.

1.1 SCOPE.

This document provides:

1. Rationale explaining technical and methodology decisions made in the development of the SCAS.
2. Tutorial material and/or references on topics central to understanding the technologies involved in producing Modular Software Radio architecture and applications.
3. Background explanation of the requirements in the SCAS.

1.2 COMPLIANCE.

All compliance rationale from this entire section is covered in the SCAS.

2 OVERVIEW.

Section 2 of the SCAS contains a descriptive overview of the architecture as a framework for communications systems. Much of that descriptive material is derived from the Step 1 Architecture Definition Report (ADR) which appears in Appendix C to the SRD. Technical details and requirements of the architecture are contained in sections 3 through 5 of the SCAS and the corresponding rationales are contained in sections 3 through 5 of this document.

2.1 CORE FRAMEWORK CONCEPT.

Core Framework is an architectural concept that defines a set of open application-layer interfaces and services that provide an abstraction of the underlying software and hardware layers for software application designers. All interfaces defined in section 3.1.3 of the SCAS were developed as part of the JTRS CF. Some of these are implemented by the CF Core Application Services (incorrectly called a "Core Framework"); others are implemented by non-core Applications (i.e., waveforms and logical devices for manufacturers' hardware devices.). Core Application Services (CAS) are identified in Figure 2-1 in solid blue.

Organization of SCAS section 3.1.3 is intended to present the CF interfaces in logical groupings. It is not organized by requirement nor implementation, "CF vs. waveform application" and the groupings are independent of test method. The groupings selected are:

1. Base Application Interfaces (*Port*, *LifeCycle*, *TestableObject*, *PropertySet, PortSupplier, ResourceFactory,* and *Resource*) are used by all waveform applications and waveform developers. SCAS lists the required behavior for these interfaces
2. Framework Control Interfaces (*Application*, *ApplicationFactory, DomainManager*, *Device*, *LoadableDevice, ExecutableDevice*, *AggregateDevice* and *DeviceManager*) provides control of the system. These control interfaces allow the deployment of waveform applications. This includes waveform setup and tear down. They are provided by a CF provider or device developer.
3. Framework Services Interfaces support both core and non-core applications (*File, FileSystem, FileManager, and Log*) provided by CF provider and device developers.
4. A Domain Profile describes the properties of hardware devices (Device Profile) and software components (Software Profile) in the system.

The Domain Profile supports the combination of resources to create applications. Device Profile and Software Profile files utilize an XML vocabulary to describe specific characteristics of either software or device components with regard to their interfaces, functional capabilities, logical location, inter-dependencies, deployment properties and other pertinent parameters.

Waveform developers implement Base Application Interfaces. They do not implement the *Application* class, which is used for control, or any other Core Application Services.

2.2 SCAS COMPLIANCE.

For SCAS compliance certification, vendors must demonstrate SCAS compliance of their products.

Hardware vendors develop "hardware devices." *Devices* (Logical) are created for "hardware devices." The *Device* unit tests being developed for JTRS 2A can be used to validate device implementations against the SCAS.

Companies developing waveform applications must certify that their waveforms are SCAS compliant.

Companies providing Core Application Services must certify that their Core Framework Services (File Services, *DeviceManager*, if included, *DomainManager*, *Application*, *ApplicationFactory*) are SCAS compliant.

As a result of using SCAS compliant hardware devices, *Devices* and *Applications*, a company building a system can procure Core Application Services from company A, hardware devices and/or logical *Devices* from company B and Waveform Applications from company C while maintaining SCAS compliance.

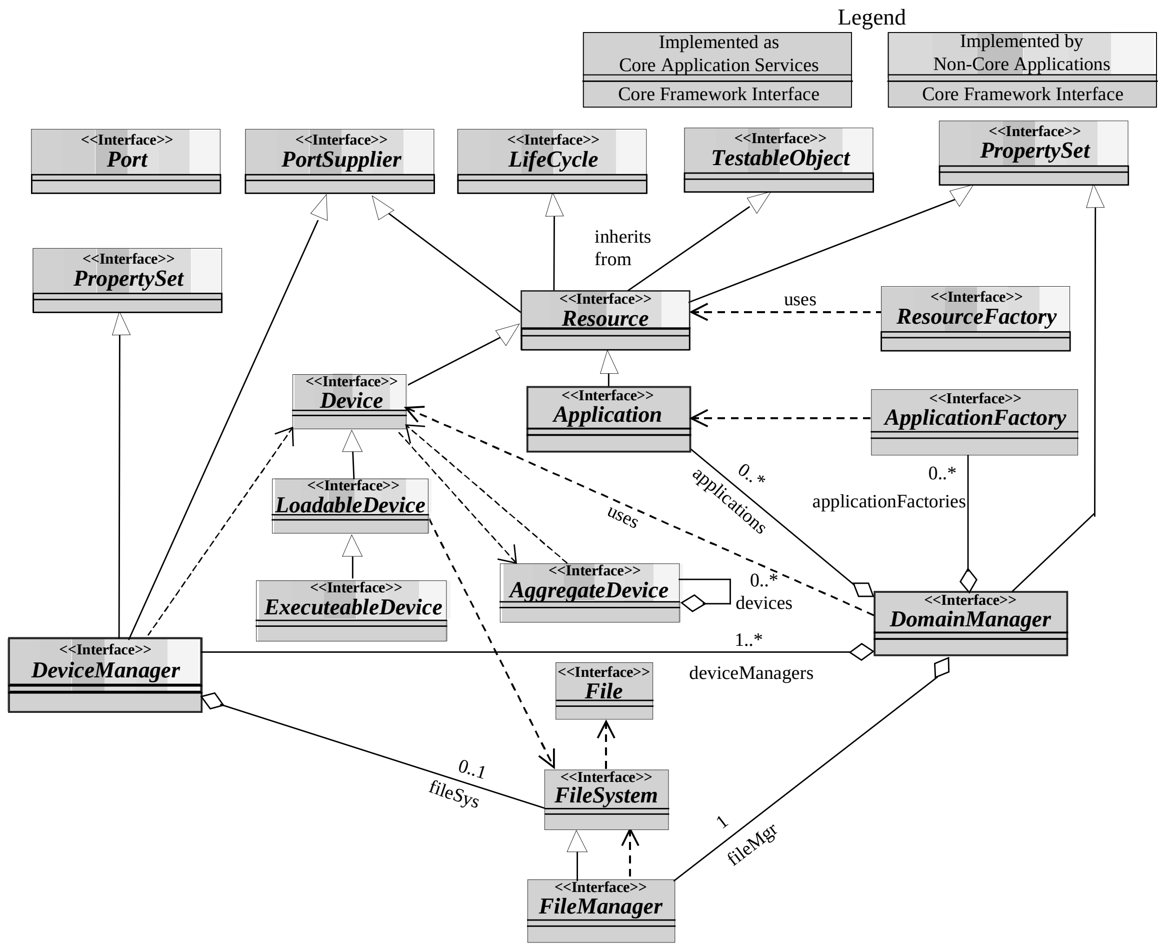

Legend
Implemented as Core Application Services
Core Framework Interface
Implemented by Non-Core Applications
Core Framework Interface
<<Interface>> Port
<<Interface>> PortSupplier
<<Interface>> LifeCycle
<<Interface>> TestableObject
<<Interface>> PropertySet
<<Interface>> PropertySet
inherits from
<<Interface>> Resource
uses
<<Interface>> ResourceFactory
<<Interface>> Device
<<Interface>> Application
<<Interface>> ApplicationFactory
<<Interface>> LoadableDevice
0..* applications
uses
0..* applicationFactories
<<Interface>> ExecuteableDevice
<<Interface>> AggregateDevice
0..* devices
<<Interface>> DomainManager
<<Interface>> DeviceManager
1..* deviceManagers
<<Interface>> File
0..1 fileSys
<<Interface>> FileSystem
1 fileMgr
<<Interface>> FileManager

Figure 2-1. Core Framework Relationships

3 SOFTWARE ARCHITECTURE DEFINITION.

3.1 OPERATING ENVIRONMENT.

This section contains the rationale based on the requirements of the operating system (OS), middleware, and the CF interfaces and operations that comprise the operating environment (OE).

Paragraphs in section three of the SCAS often times address more than one requirement or concept at a time. Due to this, section three in this document reiterates some of the pertinent paragraphs from the SCAS (shown in bolded font). Rationale for each concept is then presented individually where appropriate.

There is also some rationale that does not fit exactly in a paragraph. In these cases, a next-level paragraph has been added.

3.1.1 Operating System.

The processing environment and the functions performed in the architecture impose differing constraints on the architecture. An SCAS application environment profile (AEP) was defined to support portability of waveforms, scalability of the architecture, and commercial viability. POSIX services are used as a basis for this profile. See Figure 3-1. Notional Relationship of OE and Application to the SCAS AEP. The OS provides the services designated as mandatory by the AEP defined in Appendix B of the SCAS. The OS is not limited to providing the services designated as mandatory by the profile. The Common Object Request Broker Architecture (CORBA) Object Request Broker (ORB) and the CF are not limited to using the services designated as mandatory by the profile.

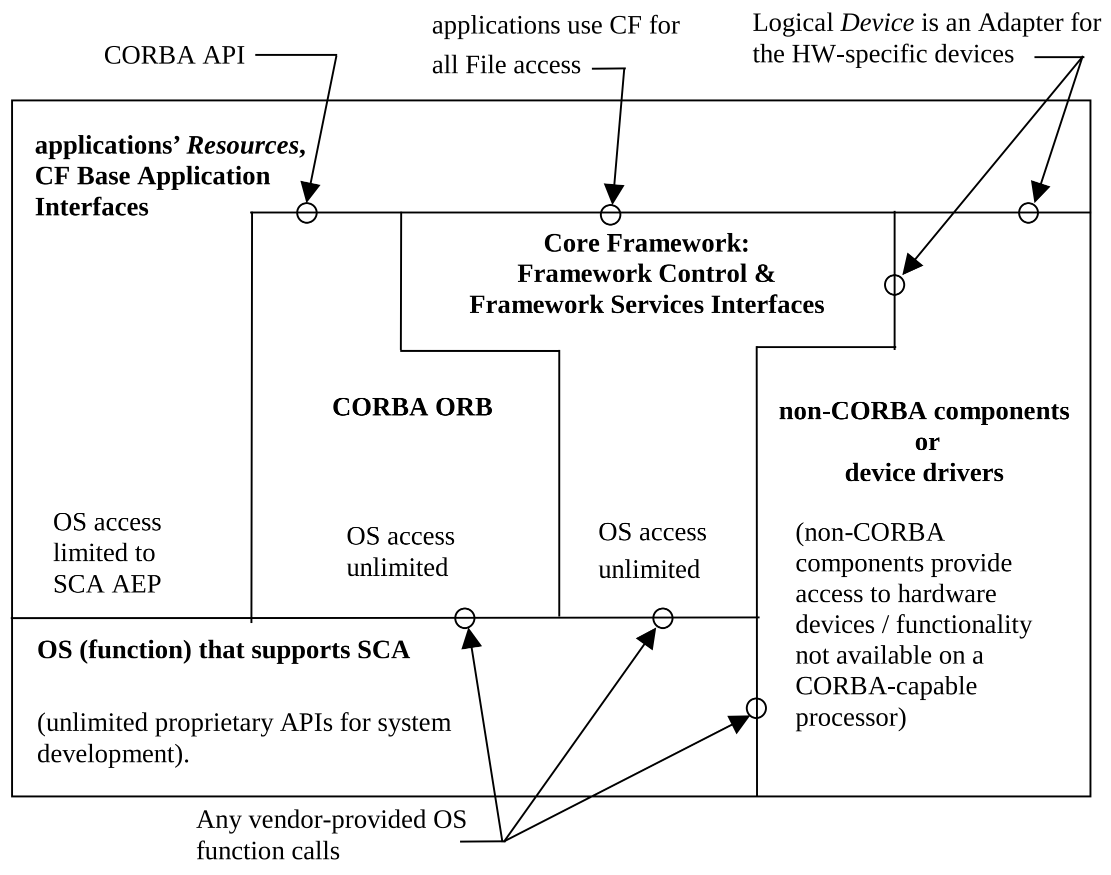

CORBA API
applications use CF for all File access
Logical *Device* is an Adapter for the HW-specific devices
applications' *Resources*, CF Base Application Interfaces
Core Framework: Framework Control & Framework Services Interfaces
CORBA ORB
non-CORBA components or device drivers
OS access limited to SCA AEP
OS access unlimited
OS access unlimited
(non-CORBA components provide access to hardware devices / functionality not available on a CORBA-capable processor)
OS (function) that supports SCA
(unlimited proprietary APIs for system development).
Any vendor-provided OS function calls

Figure 3-1. Notional Relationship of OE and Application to the SCAS AEP

3.1.1.1 POSIX.

A key requirement for JTRS waveforms is portability. For waveforms to be portable, they must conform to a standard set of interfaces to the operating system services. The set of POSIX standards established by the IEEE defines such standard interfaces. POSIX is the only existing standard for real-time operating systems (RTOS) interfaces, that is implemented wholly or in part by multiple operating systems and was therefore chosen as the basis for standardization of the SCAS OS interfaces.

3.1.1.2 POSIX 1003.13.

The POSIX 1003.13 standard defines four Application Environment Profiles (AEP), PSE-51, PSE-52, PSE-53 and PSE-54. Each application profile defines the interfaces that must be implemented for the profile in terms of units of functionality in each of the POSIX.1, POSIX.1b, POSIX.1c and POSIX.5b standards. Each profile is designed to fit an application model. A brief description of the profiles follows.

3.1.1.2.1 Minimal Real-time System Profile (PSE-51).

This profile is a single process profile with no asynchronous or file Input/Ouput (I/O) specified. Systems that implement this profile are typically embedded controllers.

3.1.1.2.2 Real-time Controller System Profile (PSE-52).

This profile is a single process profile that adds asynchronous and file I/O to PSE-51. Systems that implement this profile are typically embedded controllers.

3.1.1.2.3 Dedicated Real-time System Profile (PSE-53).

This profile adds multi-process capability to PSE-51.

3.1.1.2.4 Multi-Purpose Real-time System Profile (PSE-54).

This profile includes all the capabilities of the other three profiles and adds multi-user capabilities. The functionality includes all of POSIX.1, POSIX.1b and POSIX.1c and/or POSIX.5b.

3.1.1.3 Choosing a Profile.

One factor to consider when choosing a profile is that POSIX 1003.13 assumes no abstraction layer or middleware. The profiles are written such that for a given model applications make direct calls to the OS to use certain services. With the existence of CORBA middleware and the CF abstraction layer in the architecture, some of the burden of achieving waveform portability can be shifted from the profile to those other elements.

An OS may be chosen for a given implementation of the SCAS and configured (read scaled) to accommodate a specific domain. Therefore, the SCAS must define a minimum set of POSIX functionality that the OS must provide in order to preserve waveform portability across domains while still providing the functionality required by waveforms. In addition the SCAS must in no way prohibit operating systems that implement additional features as they may provide needed or desired functionality for a given domain. Additional functionality provided, by the OS, must either be abstracted by the Core Framework or be transparent. In this way the architecture is allowed to scale from resource limited domains to full blown fixed installations. By limiting the applications to the SCAS defined profile, then maximum waveform portability is ensured.

3.1.1.4 PSE-SCAS.

The definition of a SCAS POSIX profile must allow a vendor of SCAS compliant radios to choose an operating system based on factors such as domain specific needs, features, cost, tool set and other factors. Since most operating systems, with guaranteed real time behavior, are of the single process type, the PSE-52 AEP defined in POSIX 1003.13 was chosen as a starting point. Although the PSE-52 profile is a single-process AEP, the SCAS modifies it and uses it in such a way as to allow multi-process, multi-user operating systems that may conform to PSE-54. In addition, PSE-52 includes file and file system capability which the SCAS needs. This modified AEP is called PSE-SCAS. Since this profile does not require a particular memory model it allows for different memory protection schemes such as the process model as typified by UNIX an its derivatives and protection domains which have been implemented in experimental operating systems such as Pebble[1] and Opal[2].

VxWorks is the most popular real-time operating system and a good example of a single process operating system. The profile therefore was tailored to allow VxWorks which may be suitable for certain domains.

3.1.1.4.1 Application conformance to PSE-SCAS.

One of the goals, of the SCAS, is to achieve maximum portability for applications. If an OS provides interfaces and functionality not specified in the AEP and an application makes use of the functionality by making direct calls to the OS, then that application would have to be modified to port it to a platform with an OS that did not support the additional functionality. To achieve the goal of maximum portability, applications must therefore be limited to the PSE-SCAS AEP.

3.1.1.4.2 OS conformance to PSE-SCAS.

Application's adhering to the PSE-SCAS AEP must be supported by an OS providing the AEP interfaces.The difference is that the AEP is a maximum for an application and a minimum for an OS. That is, the OS in an SCAS compliant platform must at a minimum provide the interfaces and functionality that the AEP specifies. The OS can provide additional functionality either through vendor specific interfaces or through additional POSIX interfaces including full PSE-54 conformance. The term OS, in this context, includes third party vendor provided interfaces and functionality. For instance, software from a third party vendor can provide the pthread interfaces specified in the AEP to abstract OS vendor specific interfaces.

3.1.1.4.3 CF conformance to PSE-SCAS.

The CF is not restricted to the PSE-SCAS. Although it may be desirable for the CF to be portable across platforms and operating systems, it is not necessary. In fact, to achieve maximum portability of applications across operating systems, it may be necessary to give up some portability in the CF to achieve the more important goal of application portability.

3.1.1.4.4 Deviations from PSE-52.

3.1.1.4.4.1 POSIX.1.

3.1.1.4.4.1.1 POSIX_SINGLE_PROCESS Functions.

---

[1] The Pebble Component-Based Operating System

Eran Grabber, Christopher Small, John Bruno, Jose' Brustoloni and Avi Silberschatz

http://www.bell-labs.com/project/pebble/

[2] Sharing and Protection in a Single-Address-Space Operating System

Jeffrey S. Chase, Henry M. Levy, Michael J. Feely, and Edward D. Lazowska

ACM Transactions on Computer Systems, 12(4), November 1994

3.1.1.4.4.1.1.1 uname.

The uname function retrieves the system name, node name, OS release and version and machine (hardware identifier). This function was omitted from PSE-SCAS for the following reasons:

1. It is not consistently implemented across operating systems.
2. Identity is established by the CF::Device element, which can use OS specific calls.

3.1.1.4.4.1.1.2 sysconf.

The sysconf function provides a method for an application to determine the current value of a configurable system limit or option. This function was omitted from the profile for the following reasons:

1. It is not consistently implemented across operating systems.
2. The SCAS explicitly specifies which features must be supported and therefore dynamic reconfiguration based on the values of sysconf flags is not an option.
3. Many of the sysconf limit values pertain to the file system and the SCAS standardizes the file system by requiring that all file access go through the CF.
4. Other limit values pertain to some resource capacity that may be addressed by the CF::Device.

3.1.1.4.4.1.2 POSIX_MULTI_PROCESS Functions.

3.1.1.4.4.1.2.1 setlocale.

This function is not a strictly multi-process function and was deemed to be of some usefulness for foreign radios.

3.1.1.4.4.1.3 POSIX_SIGNALS Functions.

3.1.1.4.4.1.3.1 alarm.

The alarm function is process oriented. It sends signals to a process and not a thread. Since some RTOSs do not implement processes they do not implement alarm. The alarm time granularity is seconds which is of limited utility in real time environment such as a radio. In addition, the functionality provided by alarm is also provided by POSIX timers which have a much higher time resolution.

3.1.1.4.4.1.3.2 sigsetjmp, siglongjmp.

These functions were omitted because they are not implemented consistently across operating systems. If the purpose is to jump from a signal handler to cleanup code, setjmp and longjmp suffice.

3.1.1.4.4.1.4 POSIX_FILE_SYSTEM Functions.

The SCAS states that although file system functions are included in the AEP, applications are prohibited from using them directly. The reasoning behind this is that the file access functions are implemented widely in operating systems and therefore there would be no penalty in including them in the profile and requiring an operating system to provide them. The CF could use these functions and be more portable.

The CF is designed to provide location transparency for file access with CORBA serving as the distribution mechanism just as remote procedure calls (RPC) do for the Sun Networked File System (NFS). To maintain transparency, applications must therefore be restricted to using the CF File System.

3.1.1.4.4.1.5 POSIX_FD_MGMT Functions.

3.1.1.4.4.1.5.1 dup.

This function is normally used for inter-process communication, (IPC), (i.e., pipes). Single address operating systems do not implement processes and the responsibility for IPC lies with the CF. This function was therefore omitted from the profile.

3.1.1.4.4.1.5.2 dup2.

This function is normally used for redirection of stdin and stdout. File I/O operations in applications are restricted to the functions provided by the CF for the reasons stated in section 3.1.1.4.4.1.4.

3.1.1.4.4.1.5.3 fcntl.

Applications are restricted to using the CF for file I/O for the reasons stated in section 3.1.1.4.4.1.4.

3.1.1.4.4.1.6 POSIX_DEVICE_IO Functions.

Adapters may need to communicate with devices directly so applications may use these functions. Applications may not use these functions for file I/O for the reasons stated in section 3.1.1.4.4.1.4.

3.1.1.4.4.1.7 POSIX_C_LANG_SUPPORT Date and Time Functions.

3.1.1.4.4.1.7.1 tzset.

The setting of the time zone is more properly a global function within a radio system or a LAN. Applications, as defined in JTRS should not be setting the time zone. The time should be set on a workstation on a LAN or through the CF (i.e., time service).

3.1.1.4.4.2 POSIX.1b.

3.1.1.4.4.2.1 _POSIX_MAPPED_FILES.

Mapped files can be used to implement IPC or to persistently store data using memory access semantics in a language. The IPC mechanism in the SCAS is CORBA, which allows for transparent distribution of applications and objects. Persistent storage should be accomplished through the CF file system.

3.1.1.4.4.2.2 _POSIX_SHARED_MEMORY_OBJECTS.

Shared memory is used to communicate between processes in a multi-process operating system without incurring the overhead of pipes. Processes may be distributed across processors. Using this capability directly prohibits location transparency of the processes. The IPC mechanism in the SCAS is CORBA, which allows for transparent distribution of applications and objects. If efficiency is a concern, a pluggable transport for CORBA that uses shared memory may be used as CORBA implementations are not restricted to the profile.

3.1.1.4.4.2.3 _POSIX_SYNCHRONIZED_IO.

3.1.1.4.4.2.3.1 fdatasync.

Applications are restricted to using the CF for file I/O for the reasons stated in section 3.1.1.4.4.1.4.

3.1.1.4.4.2.4 _POSIX_FSYNC.

3.1.1.4.4.2.4.1 fsync.

Applications are restricted to using the CF for file I/O for the reasons stated in section 3.1.1.4.4.1.4.

3.1.1.4.4.3 POSIX.1c.

3.1.1.4.4.3.1 POSIX_FILE_LOCKING functions.

Applications must use the CF for file operations. The CF has the responsibility for controlling mutually exclusive access to files.

Applications are limited to using the OS services that are designated as mandatory for the profile. Applications perform file access through the CF. (Application requirements are covered in section 3.2.)

3.1.2 Middleware & Services.

3.1.2.1 CORBA.

CORBA is used in the CF as the message passing technique for the distributed processing environment. CORBA is a cross-platform framework that can be used to standardize client/server operations when using distributed processing. Distributed processing is a fundamental aspect of the system architecture and CORBA is a widely used “Middleware” service for providing distributed processing.

3.1.2.1.1 CF Interfaces.

All CF interfaces are defined in Interface Definition Language (IDL). The CORBA protocol provides message marshalling to handle the bit packing and handshaking required for delivering the message. The SCAS IDL defines operations and attributes that serve as a contract between components.

3.1.2.2 CORBA Extensions.

The following extensions and/or services above and beyond minimumCORBA are allowed.

3.1.2.2.1 Naming Service.

The Naming Service is a CORBA extension provided by the OE. A Naming service is a required service within the OE. The service is used to retrieve *DomainManager* and application components object references. Static Stringified IORs are not allowed for application components since this would not work for multiple instantiations of an application and the application's Software Assembly Descriptor (SAD) file would not be portable.

The OMG Naming Service IDL was chosen since many CORBA vendors supply an OMG compliant Naming Service implementation and this specification is standard in industry (OMG, Java). The SCAS defines a subset of the OMG Naming Service IDL that a Naming Service implementation must provide to be SCAS compliant. A radio vendor may supply, as part of the Operating Environment, a "light weight" Naming Service meeting the minimum requirements set forth by the SCAS. The minimum set of operations for Naming Service is based upon the operations needed by the *ApplicationFactory* for obtaining component's object references, application components for registering their object references, and the *Application* components to destroy naming context and component object references.

The Naming Service's NameComponent structure is composed of an id-and-kind pair. The "id" element contains a string value that uniquely identifies a NameComponent. The "kind" element contains the "" (null string). A defined naming convention is necessary in order to ensure that *ApplicationFactory* and application component implementations are building the same NameComponent structure. The Interoperable Naming Service specification states the whole NameComponent structure is used for comparison, therefore if an application's component uses a different kind value then the *ApplicationFactory* would fail when performing the Naming Service resolve operation.

3.1.2.3 Log Service.

The Logger interface changes for SCAS versions 2.1 and 2.2 were initiated by CP 142 and CP 486.

CP142 (resolution incorporated into SCAS 2.1) identified the following two main issues:

1. What is the intent and operation of logger?
2. What mechanism is there for a Waveform application to understand errors or report anomalies during runtime?

These two issues resulted from ambiguities in SCAS 2.0 regarding the location of the Logger(s), which Logger can components use, how does a client get a reference to the *DomainManager's* Logger, and how is error reporting/error resolution related to the Logger. The SCAS 2.0 Logger definition provided event service like functionality by forwarding log messages to registered consumers.

During the resolution of the Logger Change Proposals (CPs), the SRT investigated the use of the OMG Log service. With the JTRS goal of commercial standardization of the JTRS interfaces in mind, the SRT performed a detailed investigation and analysis of the OMG Log service. The investigation and analysis led to the creation of a lightweight version of the OMG Log service. Each interface attribute and operation of the OMG Log service was analyzed to determine if it provided required JTRS Log functionality. The required attributes and operations resulting from the analysis were utilized to create the lightweight JTRS Log interface. The *Log* forwarding capability was removed during this CP's resolution. The forwarding capability was deemed to be an Event Service function. The resolution of this CP also incorporated configurable connections in *Application* XML to connect *Log* objects to *Log* producers.

CP486 (incorporated into SCAS 2.2) contained two main issues with the *Log* and log record identification. The issues were:

1. It was not clear how the log names were produced.
2. It appeared that there was no way to correlate a log record with the correct log producer, given the current definition of the *Log* Producer ID.

The resolution of this CP resulted in the *Log* Producer ID being formed from the components identifier. Each component in the JTRS system contains a unique identifier. The CP SRT determined the producers name attribute required no change. Each *Log* producer produces an implementation dependent name for logging purposes. The *Log* Producer ID provides the unique identification of the *Log* producer.

3.1.2.3.1 Use of Log Service.

No further explanation is necessary.

3.1.2.3.2 LogService Module.

Since the implementation of the LogService is optional, components that produce log records are required to implement configure properties. They are used to configure the component to produce a user-defined set of log records. CF components that are required to write log records are also required to account for the absence of a log service and otherwise operate normally.

The Log Service has the following requirements imposed on the JTRS radio system:

1. Log producers implements a configure property with an Id of "PRODUCER_LOG_LEVEL".
2. The type of this property is a LogLevelSequence.
3. Log levels which are not in the LogLevelSequence is disabled.
4. Log producers implements a property with an Id of "PRODUCER_LOG_ID".
5. The log producer uses the value of the PRODUCER_LOG_ID property in the ProducerID field of the ProducerLogRecord.
6. Log producers operates normally in the case where the connections to a Log are nil or an invalid reference.
7. Log producers outputs only those log records that correspond to enabled LogLevelType values.

3.1.2.3.3 Log.

A Log is utilized by CF and CORBA capable application components to store informational messages. These informational messages are referred to as 'log records' in this document. The interface provides operations for writing log records to a Log, retrieving LogRecords from a Log, controlling of a Log, and getting the status of a Log.

3.1.2.4 CORBA Event Service and Standard Events.

No further explanation required.

3.1.2.4.1 CORBA Event Service.

CORBA Event Service was added to SCAS v2.2. It is based on the OMG Event Service: OMG Document formal/01-03-01:Event Service, v1.1. OMG Event Service IDL: OMG Document formal/01-03-01: Event Service IDL, v1.1.

The OMG Event Service IDL was chosen since many CORBA vendors supply a full implementation of an OMG compliant Event Service and this specification is standard in industry (OMG, Java). The SCAS defines a minimum operation set for an Event Service implementation that can be provided and still be SCAS compliant. The minimum operation set is based upon the push model and un-typed Event Service. Most CORBA Event Service implementations are based upon the un-typed approach instead of the typed event service. The push model is the approach used in the SCAS API Supplement for asynchronous data communication among application components. The push model also ensures consistency with SCAS implementations by ensuring interoperability of the *Devices* and HCIs (or other domain management clients) with the *DomainManager*, *Application* and *ApplicationFactory* components, and between *Applications* and their components along SCAS implementations.

The Notification Event Service was not chosen because the use case analysis performed to date did not warrant the need for this capability (e.g., filters, Quality of Service, priority, sequences). Also, the Real-Time Notification Service is still not a standard within the OMG. The systematic process of eliminating the various features that the notification service offered and seeing what was left determined the Event Service. Since the OMG Notification Service is derived from an upward compatibility with the OMG Event Service Specification, the SCAS could migrate to this service as use cases are developed that warrant other OMG Event Service Specification behaviors.

The design goals that led to the SCAS defined CORBA Event Service definition were:

1. Lightweight design and implementation. The goal was not to over burden the small foot-print system and not under supply the larger systems of needed options and features
2. Adaptable to the established notification and events services currently used in a CORBA environment.
3. Simple design with minimal bells and whistles.

The CORBA Event Service provides the capability to provide de-coupled communication between producers and consumers. It is not intended for real time system control. The SCAS API Supplement is where the real-time APIs should be built for application components. The CORBA Event Service provides asynchronous notification of non-critical changes to the JTRS system. The CORBA Event Service is optional for application developers but mandatory for *Device*, *Application*, *ApplicationFactory*, and *DomainManager* developers. This allows for efficient implementations of Domain Management and HCI implementations, and allows HCIs to be aware of changes within the environment. An application developer is responsible for developing waveform components and an *Application* developer is responsible for implementing the *Application* interface.

A radio vendor may supply, as part of the Operating Environment, a “light weight” Event Service which meets the minimum requirements set forth by the SCAS. The minimum set of operations for CORBA Event Service is based upon the operations needed by components for pushing events to the event channel and for components receiving events from the event channel. The CORBA Event Service abstracted away the Event Channel from consumers and producers, which simplifies the behavior of components by allowing the components developers to concentrate on their business model instead of the administration behavior of an event channel. The Event Channel and administrator interfaces were left out of the SCAS, so SCAS implementations would have a choice of implementing their own event channel service or using a COTS event service. In either case, the Event Service would be compatible with components using the CORBA Event Service.

The CORBA Event Service is required in the JTRS operating environment (OE). The CORBA Event Service supports the Push interfaces (Push Consumer and Push Supplier) of the CosEventComm CORBA module as described in OMG Document formal/01-03-01:Event Service, v1.1. The CosEventComm PushSupplier interface is used byproducers for generating events and the CosEventComm Push Consumer is used by consumers for receiving events.

For connections established for a CORBA Event Service’s event channel, the *ApplicationFactory create* operation connects a CosEventComm PushConsumer or PushSupplier object to the event channel as specified in the SAD’s *domainfinder* element. If the event channel does not exist, the *create* operation creates the event channel.

For connections established for a CORBA Event Service’s event channel, the *registerDeviceManager* operation connects a CosEventComm PushConsumer or PushSupplier object to the event channel as specified in the DCD’s *domainfinder* element. If the event channel does not exist, the *registerDeviceManager* operation creates the event channel.

The *unregisterDeviceManager* operation disconnect consumers and producers (e.g., *Devices*, *Log*, *DeviceManager*, etc.) from a CORBA Event Service event channel based upon the DCD. The *unregisterDeviceManager* operation may destroy the CORBA Event Service event channel when no more consumers and producers are connected to it.

The *Application releaseObject* operation disconnects consumers and producers from a CORBA Event Service’s event channel based upon the SAD. The *releaseObject* operation may destroy a CORBA Event Service’s event channel when no more consumers and producers are connected to it.

The CORBA Event Service was designed to work in a distributed environment. It allows multiple consumers of a single event and multiple suppliers of an event. The suppliers and consumers are not required to have knowledge of each other’s object reference. A single connection to an Event Channel provides access to all members associated with that Event Channel. The CORBA Event Service Interfaces are defined in OMG IDL. They communicate event data in generic (not typed) form, to be compatible with the OMG specification. Be aware that the Event Service does not use the features of the Notification Service.

It was decided that the OE would provide two standard event channels. These two standard events are:

- Incoming Domain Management Event Channel (“IDM_Channel”).
- OutGoing Domain Management Event Channel (“ODM_Channel”).

The Incoming Domain Management event channel is used by components (e.g., *Device* to notify of state change) within the domain to generate events that are consumed by domain management functions.

The Outgoing Domain Management event channel is used by domain clients (e.g., Human-Computer Interface (HCI)) to receive events (e.g., additions or removals from the domain) generated by domain management functions.

The *DomainManager* is responsible for creating the Incoming Domain Management and Outgoing Domain Management event channels. The OE allows application developers to setup other “non-standard” event channels.

3.1.2.4.2 StandardEvent Module.

A set of events was developed for the SCAS to standardize the event related information communicated within and from the system. These “Standard events” are structures that contain specific information about what occurred and which components were involved in the event. Currently, the SCAS only considers domain management and state change events, other system events that need to be considered for SCAS inclusion are security, QOS, alarm, flow control, and internal errors. Standard Events are NOT intended for hard real-time control. They are defined to allow clients to take appropriate action as a result of the occurrence of the event within a JTR.

The StateChangeEventType provides the capability to identify a state change for either administrative, operational, and usage states. The *Device* is given the responsibility for generating this event when its state changes, thus allowing for efficient CF domain management implementations. Application developers could use this type for their component implementations.

The DomainManagementObjectRemovedEventType is used for outgoing domain events to indicate what element has been removed from the system (domain). The type is based upon the elements that get destroyed by the *Application* or unregisters with the *DomainManager*. The DomainManagementObjectAddedEventType is used for outgoing domain events to indicate the elements that are created by the *ApplicationFactory or* registers with the *DomainManager*. This type contains the source IOR of the element object reference that was added to the domain. By supplying this source IOR, the HCI does not have to call back to the *DomainManager* to obtain this information. Both of these types allow for efficient implementations of the HCI and for the HCI to become aware asynchronously of changes to the domain (system). These types also identify the element that was added or removed from the domain.

All these event types have a producer ID and a source ID attributes. The Producer ID identities the producer who generated the event and the source ID identifies the source of the event. An example of this is the *ApplicationFactory*, which is the producer for the

DomainManagementObjectAddedEventType that contains the source ID and IOR for the *Application* that was created by the *ApplicationFactory*.

Other formats for the events were considered such as event interfaces by inheritance or union type definition that encompasses all event types, but these were considered too complex and less efficient implementations than the simple event type definitions even with some redundant information (source and producer Ids).

3.1.3 Core Framework.

No further explanation required.

3.1.3.1 Base Application Interfaces.

Base Application Interfaces are the basic building blocks for all application components, *Applications*, and *Devices* within a JTRS.

3.1.3.1.1 *Port.*

The *CF Port* interface is used to support the Software Profile's Software Assembly Descriptor (SAD) in setting up or tearing down connections between CORBA components. *Port* has evolved from the *MessageConsumer* and *MessageProducer* interfaces of SCAS 0.3. These interfaces had evolved from the JTRS Step 1 *MessageRegistration* and *Message* interfaces. Advantages of the *Port* interface over the previous interfaces are:

1. The interface supports the connection concepts in the CORBA Components Specification where any provides port type can be connected to another uses port, as long the uses port understands the port, which it is connecting.
2. The uses port can be behave either as a push producer or a pull consumer. Likewise, the provides port type can be behave either as a push consumer or pull producer. Note in the SCAS API Supplement, only the push model is being used.
3. The implementation of a port allows for fan-in or fan-out implementations.
4. Specific interfaces can be developed for a port, thus eliminating no-op operations when an all encompassing interface is used (e.g., *Message*, *MessageConsumer*).
5. A simple and specific interface.

ApplicationFactory and *Application* implementations use the *Port* interface. Application developers implement the *Port* interface for their uses port. An application uses port must be a *Port* Type. The uses and provides ports are paired up in the Software Profile's Software Assembly Descriptor (SAD) or DCD, and are known as connection interface elements which are contained in the connections element. In the Software Profile's Software Component Descriptor(SCD), a port can only support one type of interface since the *connectPort* operation doesn't indicate the type of interface being used. The *connectPort* operation takes in a CORBA object reference and supports the CF *Port* interface that must later be narrowed to the specific interface (provides port) for the uses port. This generic operation allows *ApplicationFactory* and *Application* implementations to setup or tear down connections between any Application's CORBA software components. The connectionId parameter, for the *Port* operations, is a unique connection identifier created by the *ApplicationFactory* at the time a connection is created between uses and provides port or is the connection interface's connection ID in the SAD, if specified. This supports generic fan-in and fan-out implementations without the uses or provides ports actually knowing the specific ports to which they are connected.

3.1.3.1.2 *LifeCycle*.

The *Lifecycle* interface is a generic way of initializing and releasing a resource, application, or device within the domain. The operations for this interface came from the JTRS Step 1 *StateManagement* interface. They were extracted into a separate interface to allow for reuse by software components needing this behavior. By breaking this out from the *Resource* interface, it allows components to implement this behavior. This concept is an agreed upon concept from the Software Defined Radio (SDR) forum.

ApplicationFactory implementations use this interface to initialize an application's components after components have been deployed and their object references have been obtained.

HCI (or other domain management clients) uses this interface to release an application or a device. *Application* implementations use this operation to release an application's components.

3.1.3.1.3 *TestableObject.*

The *TestableObject* interface is a generic way of testing a *Resource*, *Application*, or *Device* components within a domain. Test descriptions, along with their results, are described in the Software Profile's Properties File by the *test* XML element.

These interface operations came from the JTRS Step 1 *StateManagement* interface which were pulled out into a separate interface, thus allowing for reuse by software components needing this behavior. This concept is an agreed upon concept from the SDR forum (SDRF).

TestableObject may be used by an HCI to test an *Application* or *Device* components within the domain. Using this interface, it would be possible to implement a generic HCI that could work for all Applications and *Devices* since this interface is based upon the XML *test* element.

TestableObject can be used for testing remotely over the air without operator intervention.

Currently there is no requirement to perform initial testing of an application's components by the *ApplicationFactory*.

3.1.3.1.4 *PortSupplier*.

The *PortSupplier* interface provides the *getPort* operation for those components (*Application*, *Resource*, *Device*, *DeviceManager*) that contain ports as described in their SCD XML file. This interface was pulled out of the *Resource* interface for components use that only needed to provide this behavior.

The *getPort* operation from *PortSupplier* supports the SAD and DCD *connections* element definitions. *GetPort* is used by the *ApplicationFactory* and *DomainManager* to retrieve uses and provides ports, in order to establish connections to services or to other components.

3.1.3.1.5 *PropertySet.*

The *PropertySet* interface is used to support the *properties* element. Properties (Id(name)/value pairs) are concepts from the CORBA Components, Telecommunications Information Networking Architecture (TINA) specifications, and CORBA property service specification. These interface operations came from the JTRS Step 1 *StateManagement* interface which were pulled out into a separate interface, thus allowing for reuse by software components needing this behavior. This concept is an agreed upon concept from the SDR forum.

ApplicationFactory implementations use this interface to initially configure an application.

PropertySet can be used by a HCI to configure an *Application, Device, DeviceManager, and DomainManager* within the domain. With this interface, it is possible to have a generic HCI based upon the properties XML to configure and query those components.

PropertySet interface also provides the capability of adding configure and query properties to a component's implementation without having to change an IDL interface definition. In this case, the properties XML file would have to be updated.

PropertySet can be used for over the air remote configuration without operator intervention. Or, query for information remotely over the air from any application within the domain.

3.1.3.1.6 *Resource.*

The *Resource* interface is based upon a minimal set of interfaces needed to initialize, configure, and control a component (e.g., an application's *Resources*, a *Device*, or an *Application). Resource* supports a set of behavior that enables *Application* delegation and teardown and consistent component deployment between *ApplicationFactory* and DeviceManager.

Resource is used by an HCI (or other domain management clients) to configure an *Application* or *Device* within the domain. With this interface, it would be possible to have a generic HCI that could work for all *Applications* and *Devices*. Figure 3-2 represents a *Resource* with two data ports and one data output port.

Each Resource in the system has a unique identifier. The reason for the unique identifier is so each component that is logging messages can be identified as the producer of the log message.

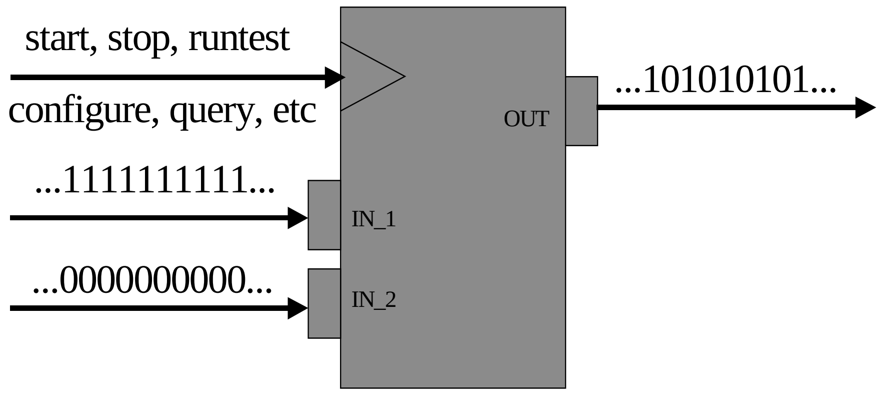


Figure 3-2. *Multiplexer* Resource

3.1.3.1.7 *ResourceFactory.*

The *ResourceFactory* interface provides a means to request the creation or destruction of a specific type of *Resource* in the domain. *ResourceFactory* was designed after the Factory Design Pattern. This interface is an optional interface that can be used by application developers.

The SPD determine the type of *Resource* that is created by the *ResourceFactory*. Properties passed to the *ResourceFactory create* operation can indicate the type of *Resource* to be created.

The *ResourceFactory* design requires an implementation trade off (e.g., security) to determine whether *Resource* instances need to be in their own individual OS Process Space or all in the same OS Process Space. A *Resource* can be executed on the same processor multiple times or it can be created multiple times using the *ResourceFactory*.

The *ResourceFactory* keeps track of the number of times the *Resource* has been referenced by clients using the *Resource*. As clients release their reference to the *Resource* the factory destroys the *Resource* when there are no more references from any client. This feature would be beneficial for managing *Resources* that are used by multiple applications and eliminates the constant redeployment of the shared components across waveform boundaries.

3.1.3.2 Framework Control Interfaces.

In the problem space, a Domain consists of software (SW) Applications (installed services), Nodes (and their devices, file systems, and services) File Manager, Event Channel, and SW Application Instances, as depicted in figure 3-3.

A Node is an abstraction of the computing node, that allows the control and monitoring of the node resources (Central Processing Unit (CPU), processes, memory and file systems), implements the OE, executes a *DeviceManager* and installed OE & CF services (e.g., CORBA Naming Service, FileSystem, *Log*, Event Service, etc.).

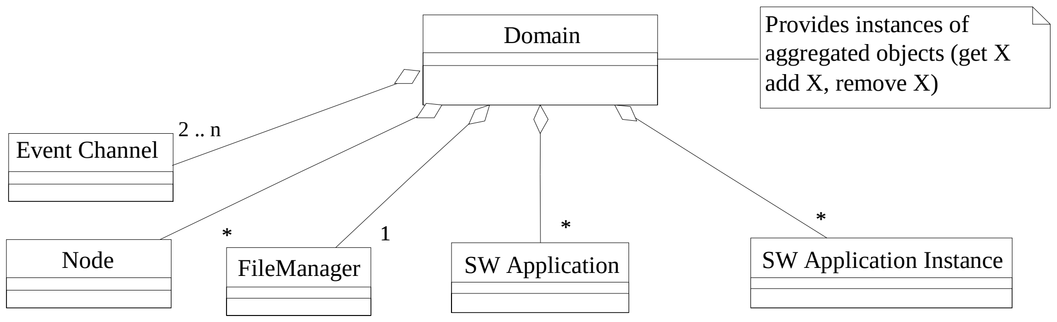


Figure 3-3. Domain In Problem Space

The services provided by a Domain (*DomainManager*) are:

1. Installing and uninstalling application software (*ApplicationFactory*) onto the domain's file manager.
2. Retrieving Nodes (*DeviceManager*), SW Applications, and SW Application Instances.

3. Creating, Terminating, and Controlling SW *Application* Instances.
4. Registering and unregistering Nodes (*DeviceManager*) along with their *Devices* and services.
5. Registering and unregistering to the event channels. In the Domain there are two event channels by default. Outgoing Domain Event Channel for letting client become aware of domain changes and Incoming Domain Event Channel for letting the Domain become aware changes within the domain.

These Domain services are mapped to the following CF interfaces:

1. *DomainManager* provides the services for retrieving, installing/registering, and uninstalling/unregistering Domain elements.
2. *Application* provides the services for terminating and controlling the SW Application Instance.
3. *ApplicationFactory* provides the services for the creation of an *Application*. For each SW *Application* that is installed in the Domain, an *ApplicationFactory* object is created that is used for deploying the application within the Domain.
4. *DeviceManager* provides the services for managing a node.
5. *Device* interfaces (*Device*, *LoadableDevice*, *ExecutableDevice*, *AggregateDevice*) provides the services for managing hardware devices.

3.1.3.2.1 *Application.*

From an end-user perspective, software applications (services, waveforms, etc.) are the primary functional element of a JTRS radio. Since Users need to control the lifespan of applications, configure them, and control their behavior, the *Application* interface provides the necessary operations for managing application lifecycle, state, and behavior. Having a generic *Application* abstraction supports general purpose user interface components for controlling a JTRS radio.

An *Application* has characteristics very similar to a *Resource*. For example, an application has properties and communicates through ports. Therefore, the *Application* interface is derived from the *Resource* interface. In general, the implementation of an *Application* is comprised of a set of *Resource* component implementations and their interconnections. In addition, the *Application* interface provides the filename containing the eXtensible Markup Language (XML) SAD profile information that describes its internal structure and the characteristics of the software assembly. The Application interface extended the Resource interface by adding deployment information (components associated with what devices, naming context names for components, etc.) of how the software assembly got deployed. The *Application* interface provides a standard interface for all applications within the domain, and one common implementation (although each CAS may be different) within the domain for all applications Application developers don't have to develop code to tear down their application and to behave according to the *Application's* software profile (SAD). The *Application* behaves as the proxy for the instantiated software assembly. This concept is similar to Enterprise Java Beans (EJB). The *Application* delegates its *Resource* operations to a *Resource* component that has been identified as the assembly controller for the software assembly. Since the *Application* interface is derived from *Resource* and an assembly control is a Resource this makes the delegation very simple and elegant. The design and implementation of the assembly is totally under the control of an application developer without the need to worry about deployment management behavior.

The *Application's* ports provide the capability of connecting the *Application* up to other components such as an Application. The ports can also be used to provide additional command/control and/or status interfaces for an Application. The ports for an Application are flexible allowing developers to determine which of their Application's components' ports should be made visible by the SAD definition. The port object references given out are the application's components' port object references thus avoiding the *Application* proxy object, which provides for an efficient implementation.

The *Application* sends out notification by the Outgoing Domain event channel of when an *Application* is destroyed. This allows for clients (HCI) to become immediately aware of an *Application* no longer available.

3.1.3.2.2 *ApplicationFactory.*

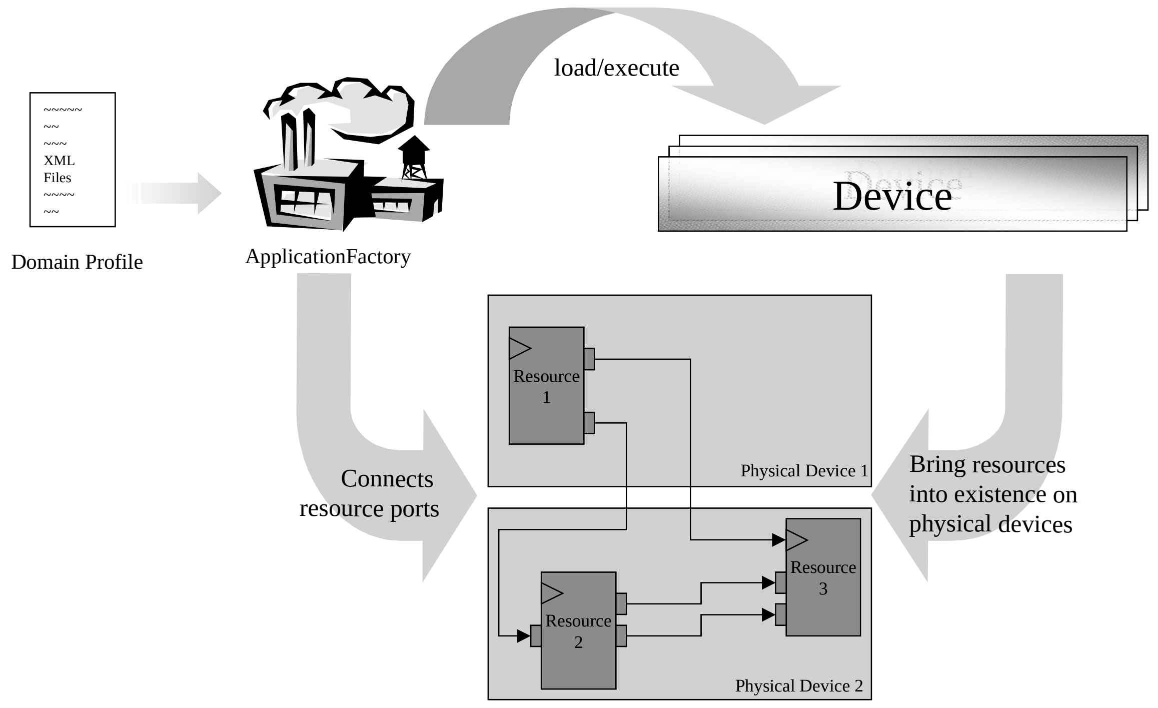
Although the *Application* interface provides the operations for controlling most of an application's lifecycle, it cannot be responsible for the actual creation of an application (since an executable instance doesn't exist yet). For a user to start an application, they must be aware of the types of applications available in the radio and then be able to command the creation of a new instance of a selected application type.

The *ApplicationFactory* provides an interface to request the creation of a specific type of application in the domain. This abstraction allows a user interface to manipulate a group of application types as objects. The user (through the user interface) can create an application instance, provide it with pre-selected devices, and specify its configuration properties. The interface also provides a user-friendly name for the application type, identifier, and provides access to the software profile (SAD) used for creating applications of that type. The identifier corresponds to the SAD ID and is used for logging message and outgoing Domain Management events, which allow identification of the *ApplicationFactory* generating events and logging messages. Figure 3-4, Example *ApplicationFactory*, illustrates how the *ApplicationFactory* interface is used to bring a new *Application* instance into existence.

The *ApplicationFactory* interface was designed based on the Factory Design Pattern. There is one *ApplicationFactory* object for each type of application and one implementation for all *ApplicationFactory* objects. Each *ApplicationFactory* object has its own instance variables based upon the same servant class definition and therefore is its own unique instance.

The *ApplicationFactory* interface provides a standard interface for creating applications within the domain, and one common implementation (although each CAS may be different) within the domain for creating applications. The *ApplicationFactory* creates a CORBA object implementing the *Application* interface for each application created. Each application developer doesn't have to develop code to parse their own software profiles and retrieve domain profile (devices, etc.) to create their application within a domain. An *ApplicationFactory* CORBA object (implementing the *ApplicationFactory* interface) is created for each different Software Assembly Descriptor (SAD) XML file installed in the domain. The *ApplicationFactory* may also be used for manual or automatic application creation. For automatic applications start up, the application software profiles (SADs) have to already exist within the domain. An application can be one or more resources, where the same resource components may be referenced by multiple/different application software profiles (SADs), thus allowing aggregate applications to be formed up. The *ApplicationFactory* also provides the mechanism for future application creation control (e.g., disable, enable). This can be done by extending the *Application* interface with more control behavior.

The *ApplicationFactory* forms Naming Contexts, which application's components use to place their CORBA object references. This provides the capability of instantiating an application multiple times. Each application' component instantiation uses the same code but a different naming context CORBA object in which to place their object references. Likewise the *ApplicationFactory* creates unique identifiers for application' s *Resource* components. This eliminates conflict between multiple instantiations of the same application. This allows log messages produced by an application's components to be uniquely identified with a component within an application's instance.



~~~~~
~~
~~~
XML
Files
~~~~
~~
Domain Profile
ApplicationFactory
load/execute
Device
Resource 1
Physical Device 1
Connects resource ports
Bring resources into existence on physical devices
Resource 3
Resource 2
Physical Device 2

**Figure 3-4. Example *ApplicationFactory***

## 3.1.3.2.3 ***DomainManager*.**

The *DomainManager* provides the repository for the elements within the system (or domain). These elements are the installed application, application instances, *DeviceManagers* (nodes and their devices and services) and event channels.

The *DomainManager* provides operations for the elements to register themselves. This registration supports startup behavior and dynamic behavior of elements being added or removed from the system. The registration technique optimizes the operation of the *DomainManager* because it does have to expend processor resources polling for new elements. The selection of a registration strategy is consistent with industry techniques such as TINA specifications.
~~~~~

When elements registered to the *DomainManager*, the *DomainManager* uses the elements' XML profiles (SAD, SPD, DCD) to obtain their deployment characteristics. As *DeviceManagers*, *Devices*, and services registered to the *DomainManager*, the *DomainManager* establishes connections to services for these elements instead of *DeviceManagers*. This technique provides the most efficient technique since the *DomainManager* knows when services become available. Connections established for services are for the *Log* and event channels.

The *DomainManager* sets up the Incoming Domain and Outgoing Domain event channels. This allows for efficient implementations of the *DomainManager* of knowing when Devices' state changes. It also allows for asynchronous notification of system changes to the outside clients (HCI).

The *DomainManager* receives information from various elements in the architecture, including all the *DeviceManagers* (DCD Profiles, *Device* SPD Profiles), and the Install applications (SAD Profiles).

3.1.3.2.4 *Device.*

A logical *Device* is a functional abstraction for a hardware device and provides the following attributes and operations:

1. Software Profile Attribute – This Software Package Descriptor (SPD) XML profile defines the logical *Device* capabilities (data/command uses and provides ports, configure and query properties, capacity properties, status properties, etc.). The use of the SPD allows a generic HMI to be written to interface with any vendor's *Device* and to be used by a *DomainManager* and *ApplicationFactory* for deployment on components onto devices.
2. State Management & Status Attributes – This information describes the administrative, usage, and operational states of the device. . Devices have state since they are viewed as managed resources within the JTRS. This information is based upon the X.731 INFORMATION TECHNOLOGY - OPEN SYSTEMS INTERCONNECTION - SYSTEMS MANAGEMENT: STATE MANAGEMENT FUNCTION. The identical text is also published as ISO/IEC International Standard 10164-2.
3. Capacity Operations **-** In order to use a device, certain capacities (e.g., memory, performance, etc.) must be obtained from the *Device*. The capacity properties vary among devices and are described in the Software Profile. A device may have multiple allocatable capacities, each having its own unique capacity model.

The *Device* interface provides basic definition for all logical *Devices* in a JTRS. The *Device* interface by itself can be used to implement Input/Output types of *Devices* (e.g., audio, serial, 1553, etc.). The device interfaces are for the implementation and management of logical devices within the domain. The devices within the domain can be simple devices with no loadable, executable, or aggregate devices behavior, or a device can be a combination of these behaviors. The device interfaces are *Device, LoadableDevice, ExecutableDevice, and AggregateDevice*.

Since the *Device* interface merely provides an abstraction of hardware, it may be the case that many *Devices* can exist per physical hardware element. A device such as a programmable modem could present itself as both a Code Division Multi-Access (CDMA) and a Time Division Multiple Access (TDMA) modem *Device* simultaneously.

This abstraction also allows physical devices to be emulated. For example, a *Device* such as a printer could be emulated on a general-purpose processor. Instead of providing a printed page of the information sent to it, the printer *Device* could present the page graphically on a display.

3.1.3.2.5 *LoadableDevice.*

The *LoadableDevice* interface extends the *Device* interface by adding software loading and unloading behavior for a particular device. Some devices within a JTRS only support loadable behavior; this interface provides the mechanism for loading software onto these devices. The actual implementation of this interface vary depending on the underlying OE. The implementation may behave as a cross-loader to another device or loader directly onto the device. There are different types of load (kernel module, driver, shared library, executable) that can be performed by the load operation. These types are based upon most OS capabilities.

3.1.3.2.6 *ExecutableDevice.*

The *ExecutableDevice* interface extends the *LoadableDevice* interface by adding execute and terminate behavior for a device. The *ExecutableDevice* interface is usually used for devices that have OS (e,g,, VxWorks, LynxWorks, Linux, etc.) that support creation of threads/processes. The execute and terminate operations' implementation behavior vary depending on the underlying OE. The interface is generic enough to accommodate most RTOS behavior (threads or processes). The parameters for the execute operation allow for the user to have control over the stack size and priority for the thread/process creation and for user parameters to be passed to the thread/process during creation. The user parameters are id and value string pairs so they can be converted to (argc, argv) format. The execute implementation uses (argc, argv) format for executable parameters to promote portability of waveform's components and this is the format used in POSIX.

3.1.3.2.7 *AggregateDevice.*

This interface provides the mechanism to associate a *Device* with another *Device*. The *AggregateDevice* interface provides aggregate behavior that can be used to add and remove *Devices* from an aggregate *Device*. This aggregate relationship can be very tightly coupled or loosely coupled. Tightly coupled is a composition relationship where the *Device* cannot exist without the other associated *Device* and cannot be associated with another *Device*. Loosely coupled means the *Device* can exist and can be associated with another *Device*. The DeviceConfigurationDescriptor (DCD)XML file can be used to describe these relationships.

3.1.3.2.8 *DeviceManager.*

The *DeviceManager* is a service that manages a set of persistent logical *Devices* and services (e.g., *Log*, Event Service, Naming Service, etc.) for a given node within a system or domain. The *DeviceManager* interface is depicted in Figure 3-6. The *DeviceManager* is viewed as a manager of devices and services for a node. It is not viewed to be a *Device*. It no longer inherits from *Device*.

A given system is composed of a set of nodes. These nodes are associated with a *DeviceManager*. The *DeviceManager* provides the capability of simultaneously starting up logical *Devices* and services on a node at power up, thus providing a faster start up of the system and treats each node as an independent object within the system. As nodes are removed or added to the system (*DomainManager*), the set of elements belonging to a node are easily identified by the attributes of the *DeviceManager* interface.

The *DeviceManager* provides the capability of changing the characteristic of the node by its associated Device Configuration Descriptor (DCD) XML file and by its operations (services and logical devices can be added or removed).

The elements managed by a *DeviceManager* vary depending on the hardware devices associated with or contained by a node and where services are deployed within a system.

A node usually has some file system associated with it. Therefore the *DeviceManager* interface has a file system attribute. The underlying implementation for a file system may be implemented as a *FileManager* when the node consists of many file systems. This file manager can be given out as a file system since the FileManager is derived from the *FileSystem* interface.

A set of well-defined executable parameters (e.g., DEVICE_MGR_IOR, DEVICE_ID, DEVICE_LABEL, etc.) is defined to promote portability of logical *Devices* and services and consistent behavior from system to system.

The *DeviceManager* interface inherits the *PropertySet* interface in order to manage implementation properties that are described in its Software Package Descriptor (SPD) file and to change its producer log level properties. The *PortSupplier* interface inherited by the *DeviceManager* interface is used to connect services (e.g., *Log*) to the *DeviceManager*.

3.1.3.3 Framework Services Interfaces.

3.1.3.3.1 File.

The *File* interface provides a distributed file service capability that is used when accessing CF elements' profile attributes, loading and executing files, installing applications, and for application's components usage. The *File* interface abstracts away where the file object resides within the system (red-side, black-side, local, or remote).

Applications must use the CF *File* interfaces so that the location of the files is transparent to the application. The applications can access all files using the CORBA *File* interface whether the file is local or remote. This provides consistency and portability of applications.

See section 3.1.1.4.4.1.4 POSIX_FILE_SYSTEM Functions for further description of why POSIX file commands were replaced.

3.1.3.3.2 *FileSystem.*

The *FileSystem* interface provides a distributed (network) file system service capability that is used when accessing CF elements' profile attributes, loading and executing files, and installing and uninstalling applications. The *FileSystem* interface abstracts away where the file system object resides within the system (red-side, black-side, local, or remote).

The interface provides the facility of passing around a logical Network File Systems, (NFS) objects as CORBA object references within the system. The *FileSystem* provides transparent access of files across platforms. This ensures all systems have a consistent technique of distributing files within the system that is CORBA based and the security mechanisms for security bypass and access control work the same as other the CORBA objects within the system since it is CORBA based. The *FileSystem* interface also was chosen over a regular Network File Systems, (NFS) capability since this may not be resident on all nodes (platforms) within a system or available for a wide range of Operating Systems, and provide a distributed CORBA API.

The interface provides basic file system operations one would expect on file system. The behavior of these operations resembles POSIX operations.

The interface also provides a generic query operation to obtain information (e.g., usage size, available size) out the file system. This can be easily extended with out modifying the interface to retrieve other file system information.

See SCAS section 3.1.1.4.4.1.4 POSIX_FILE_SYSTEM Functions for further description of why POSIX file commands were replaced.***FileManager.***

The *FileManager* interface provides a distributed file manager service capability that is used when accessing a system's (*DomainManager*) file systems. The interface can also be used for a *DeviceManager* implementation for its file system attribute when a node has multiple file systems.

The *FileManager* interface is derived from the *FileSystem* interface so it can act like and be treated like a file system, thus allowing one to perform file system operations cross its mounted file systems. These mounted file systems are treated as directories in this case from the client perspective.

The *FileManager* also extends the *FileSystem* interface by providing mount and unmount behavior like a Network File System (NFS). This allows a file manager to be built that contains all the file systems in the system and be treated like one file system for a system. This allows one to manage the system files from one CORBA object.

Since the *FileManager* is a CORBA object the access control (HMI guard) work the same as other the CORBA objects within the system since it is CORBA based.

See SCAS section 3.1.1.4.4.1.4 POSIX_FILE_SYSTEM Functions for further description of why POSIX file commands were replaced.

3.1.3.4 Domain Profile.

The elements of the Domain Profile have been based upon the OMG's CORBA Components specification (orbos/99-07-01). The CORBA Components specification was identified as a good starting point on which to base the SCAS Domain Profile because it contained much of the functionality required by the SCAS and it was already established as an OMG draft. Many of the elements of the Domain Profile described below were derived from the corresponding Component Model and then modified to fit the needs of the SCAS. This approach was taken as it builds upon the work already completed by the OMG and does not duplicate it.

3.1.3.4.1 Software Package Descriptor.

This descriptor is derived from the CORBA Components Software Package Descriptor (SPD). The SPD is used to describe any software component (non-CORBA, CORBA, etc.). The general information about a software package consists of name, author, properties, software component and implementation elements and hardware and/or software dependencies. A Software package may contain multiple implementations (for different hardware) using the same properties and Software Component Descriptor (SCD).

The differences between the SCAS definition and CORBA Components specification is the use of cardinality of the *softpkg's* elements and references to device capacity (allocation properties). The CORBA Components specification's SPD allowed random selection and sequencing of sub-elements (*softpkg*, *implementation*, etc.), which provided no way of validation and enforcement, offers no apparent benefit, and increases the XML parsing. The SPD definition is a much more simpler and an enforceable definition.

The SCAS definition of the SPD simplifies the SPD by changing the cardinality of zero to many elements with appropriate cardinality such as zero to one and enforce required elements and the order those elements should appear. These changes make the parsing of the XML much simpler and ensure a valid definition. The concept of device capacities was added to the *implementation* element. This allows deployment behavior to use the appropriate devices that have enough capacity to execute a component. In addition some elements (propertyfile, descriptor, code, etc.) were modified (restricted) to meet the requirements of a software radio instead of an enterprise definition.

3.1.3.4.2 Software Component Descriptor.

This descriptor is derived from the CORBA Component Descriptor (CCD) file definition (interfaces, provides port, uses port, supporting interface). The SCAS components (*Resource*, *ResourceFactory*, *Device*) have CORBA capable components with well-defined interfaces and both have components that use well-defined interfaces. Like those in the CORBA Components Specification, component interfaces can be described in two ways thus providing design and implementation flexibility:

1. Supporting interfaces which are IDL interfaces inherited by the component IDL interface.
2. Provides interfaces, which are known as facets that are distinct CORBA interfaces that a component implementation supports in addition to the component's interface.

A lot of the CCD's elements (security, transaction, repository, event policy, etc.) were removed for the SCD since there was no requirements to use these elements or these elements did not make sense for a software radio. Use cases have not been developed yet to justify these elements. The CCD allowed random selection and sequencing of sub-elements (like the SPD), which provided no way of validation and enforcement, offers no apparent benefit, and increases the XML parsing. The SCD definition is a much more simpler and enforceable definition.

3.1.3.4.3 Software Assembly Descriptor.

This descriptor is derived from the CORBA Component Assembly Descriptor (CAD) file definition (partitioning, connections, component files). The Software Assembly Descriptor, (SAD), like the CORBA Components Specification CAD is used to:

1. Describe what component instances makeup an assembly.
2. Describe which components are located together.
3. Describe how to find components (e.g., naming service).
4. Describe how to connect components together.

Features added to the SAD are:

1. The required identification of a component as the main assembly controller. This simplifies the behavior of the CF *Application* that acts a proxy for the assembly controller. By having a mandated assembly controller, the *Application* proxy simply delegates its *Resource* operations to the assembly controller *Resource*.
2. The optional visible ports for a software assembly. This allows the assembly to be treated by a like component that can be connected to for its data ports or additional status and command/control ports.

The CAD definition is an all-encompassing definition for enterprise software. All possibilities are defined in the CAD. This definition was scaled back based upon the needs required of a software radio, which also helped with validation and ensuring compliance. For example, trader service was not deemed necessary at this time. Naming Service is sufficient for obtaining CORBA object references for deploying components with a JTR. Stingified IORs are not used since this would not work for instantiating an application multiple times.

Component properties definitions are done differently then in the CAD in order to ensure the same definition were being used. Property definitions are defined in the SCD and SPD. These definitions can be referenced and over-ridden with different values at component instantiation time.

The concept of a home in the CAD to obtain components is accomplished in the SAD by the CF *ResourceFactory*, which acts a home for obtaining a *Resource*. Components come into existence in two ways in the SAD, one by a *ResourceFactory* and the other by deployment of a component onto a *Device* and using Naming Service to obtained the component's object reference. Within the SAD, the same *ResourceFactory* component can be used to create up different *Resource* components.

Connections definition in the SAD is vary similar to the CAD. The SAD placed restrictions of the techniques (trader service, stringified IOR) used and added a few other techniques for obtaining object references from the domain and devices a component is associated with Since file systems and services are registered and known by the *DomainManager*, the *domainfinder* element is used to instruct the deployment behavior (*ApplicationFactory*) of object references to services needed for a component's connection.

3.1.3.4.4 Properties Descriptor.

A Properties File contains information about the properties applicable to a software package or a device package. This descriptor is derived from the CORBA Components *properties* element. The elements added to the *properties simple* element definition are:

1. Mode- identifies the access control for a property. This element is only effective when the kind is "configure". This attribute defines whether the *properties* element is "readonly", "writeonly" or "readwrite"ID – identifies an unique identifier string for a property. The name element is optional. When an ID is a meaningful value, (like a name) then the simple name attribute is not necessary. The id attribute for a *simple* property that is an allocation type is a DCE UUID value, as specified in SCAS section D.2.1. The UUID is used to ensure uniqueness of the ID within the system.

2. Enumeration (Optional) – identifies a list of enumeration's with assigned values. This allows for the definition of a label-to-value for a particular property
3. Range – identifies min and max values for a property.
4. Units – identifies the unit of measure for a property.
5. Kind – identifies the kind of property. A property can be multiple kinds or used for multiple purposes. For example, to be able to query an executable property type, the kinds would be configure (and mode = read-only) and execparam.

The CORBA Components Specification *sequence* and *struct* elements were deemed to complex. The SCAS opted for a more simpler definitions of sequence and struct types. The CORBA Components Specification *sequence* element allowed zero to many sim*p*le, *sequence*, or *struct* elements (not a mixture of them). This definition enforces that the sequence is made up of the same elements but there is no enforcement that the element definition is the same for each element within the sequence. The SCAS sequence definitions, *simplesequence* and *structsequence*, are definitions that use the same element definition for each element within the sequence. Use case analysis performed to date did not justify the need for nesting of sequence elements. Use case analysis to date based upon waveform presets justified the need for simple and struct sequence types.

A *simplesequence* element is based upon a *simple* element definition. The difference between a *simple* and *simplesequence* elements is the *simple* element only has one value where a *simplesequence* element can have a list of simple values. Each *simplesequence* value element is based upon the same *simple* definition.

A *structsequence* element is based upon a *struct* element definition. Each *structsequence structvalue* is based upon the same *struct* definition.

The CORBA Components Specification *struct* element allowed zero to many sim*p*le, *sequence*, or *struct* elements (a mixture of them). The SCAS *struct* definition is only based upon one to many simple element definitions. Use case analysis performed to date did not justify the need for the complex definition as defined in the CORBA Components Specification. Use case analysis to date based upon waveform presets justified the need for struct definitions based upon simple definitions.

The SCAS added a test element to the properties file. This element provided the mechanism of describing a test using XML.

3.1.3.4.5 Device Package Descriptor.

A Device Package Descriptor (DPD) File identifies a class of a hardware device along with its properties. The Device Package Descriptor is not derived from the CORBA component specification. It has no concept of hardware devices in its specification. Device identification information, including the device name, device class, manufacture, and model number are contained in a DPD File. The DPD is based upon the concepts in section 4 of the SCAS. The DPD allows for composition device definitions to be formed up completely in one file or in multiple files (DPD can reference other DPDs). Some elements, like *propertyfile* and *author* have the same definitions as in the SPD.

3.1.3.4.6 Device Configuration Descriptor.

This descriptor is derived from the Software Assembly Descriptor (SAD) file definition (partitioning, connections, component files). A Device Configuration Descriptor (DCD) contains information about the associations to hardware that are deployed on the node, associations to hardware devices (DPD), how to find the *DomainManager*, *DeviceManager's* SPD file, and file system names. The DCD provides the capable of starting up nodes concurrently within a system.

The DCD provides a generic mechanism for deploying logical *Devices* on a node. These logical *Devices* can be from many different vendors, which supports the needs of a JTRS. This allows *Device* implementations to be ported from one platform to another platform.

The DCD provides a generic mechanism for deploying services (e.g., *Log*, Naming Service, Event Service, etc.). The DCD also provides a system administrator the flexibility on determining what nodes within the system the services should be deployed on and how many services within a system.

The DCD also provides service connection information for logical *Devices* and *DeviceManager*.

The DCD provides the system administrator the capability of naming the *FileSystems* associated with the *DeviceManager*.

3.1.3.4.7 Profile Descriptor.

This descriptor is used to identify a file name for a SAD, SPD, DMD, or DCD. The file name is a full pathname that includes a mounted file system name. This profile XML element can be returned by a CF interface's (*Device, DeviceManager, DomainManager, Application, and ApplicationFactory*) profile attribute. The profile attribute at one time use to be a *File,* but this was changed to a string type since a device may not have a file system, therefore don't force the device to create a *File* object for a profile. The profile attribute string definition is allowed to contain either a profile file path name or the full XML definition for a SPD, SAD, DMD, and DCD. This XML definition was chosen for the profile file path name in order to give out XML information in both cases for a profile attribute.

3.1.3.4.8 *DomainManger* Configuration Descriptor.

A *DomainManager* Configuration Descriptor (DMD) contains configuration information (services and *DeviceManager's* SPD) for the *DomainManager*. The *DomainManager* Configuration Descriptor is not derived from the CORBA component specification. It has no concept of domain management in its specification.

3.1.3.5 Core Framework Base Types.

These types are declared at the CF module level because they are used by many interfaces. The DataType and Properties types are defined to support the properties XML element, and the *PropertySet* and *ExecutableDevice* interfaces and other query operations. The ErrorNumberType is based upon POSIX error numbers, which provides standard OS error codes. The ErrorNumberType is used by a number of operations (e.g., *File* services and *Device* operations) for specified exception failures.

3.2 APPLICATIONS.

3.2.1 General Application Requirements.

3.2.1.1 OS Services.

The rationale on what OS services can be used by Application are described in section 3.1.1 Operating System and its subsections.

The rationale for using *File* interfaces instead of OS file services is described in section 3.1.1.4.4.1.4 POSIX_FILE_SYSTEM Functions.

Standard termination of applications promote portability of waveform.

3.2.1.2 CORBA Services.

A standard set of CORBA Services promotes reuseability of waveforms across *Application* and *ApplicationFactory* implementations with the use of common APIs.

3.2.1.3 CF Interfaces.

Standard interfaces allow for consistent behavior for *Application* and *ApplicationFactory* implementations by use of common APIs across waveform applications.

Standard execute parameters (e.g., Naming Context) and formats (argc, argv) promote portability of waveforms by utilizing common parameter definitions across waveform applications.

A standard interchange DTD formats based upon XML promotes portability of waveforms and consistent behavior for *Application* and *ApplicationFactory* implementations.

3.2.2 Application Interfaces.

Application components may be CORBA capable or non-CORBA capable components. CORBA capable components support the CF Base Application interfaces and can implement and use component specific interfaces for data and/or control. Non-CORBA capable components are for RF *Devices* that do not support CORBA and/or the SCAS AEP.

3.2.2.1 Service APIs.

See the SCAS API Supplement for details and requirements for Service APIs.

3.3 LOGICAL DEVICE.

The *Device* interface was broken into multiple interfaces (*Device*, *LoadableDevice, ExecutableDevice*, *AggregateDevice*) in order to clearly state the behavior of different types of devices, which allows validation of a device to be achievable. With the single *Device* interface, one had no clue of the operations that were implemented by a logical *Device* and if the *Device* was operating properly. The use of inheritance rather than provides ports was chosen to define the relationship between *Device, LoadableDevice*, and *ExecutableDevice.* IDL inheritance allows one to determine the type of *Device* by narrowing the CORBA *Device* object reference instead of checking the ports provided by a *Device* either by calling the *getPort* operation and/or checking the SCD XML profile. The port mechanism is less efficient and the inheritance relationship is the more natural way of expressing the relationships between these interfaces because each interface extends the other interface by adding more access methods (and their underlying behavior) to that provided by the base interface. The use case analysis *LoadableDevice* and *ExecutableDevice* interfaces did not make sense that these interfaces should be standalone interfaces. Since a *LoadableDevice* is a type of *Device* with loadable behavior. An *ExecutableDevice* is a loadable device with executable behavior. The aggregate behavior was removed from the *Device* interface and placed into a separate interface (*AggregateDevice*). The rationale for this was because not all devices are an aggregation or composition of devices. *Devices* that have aggregate relationships can implement this behavior as a provides port. This eliminates any no-op code for a *Device* implementation. The logical *Device* interfaces are depicted in Figure 3-7.

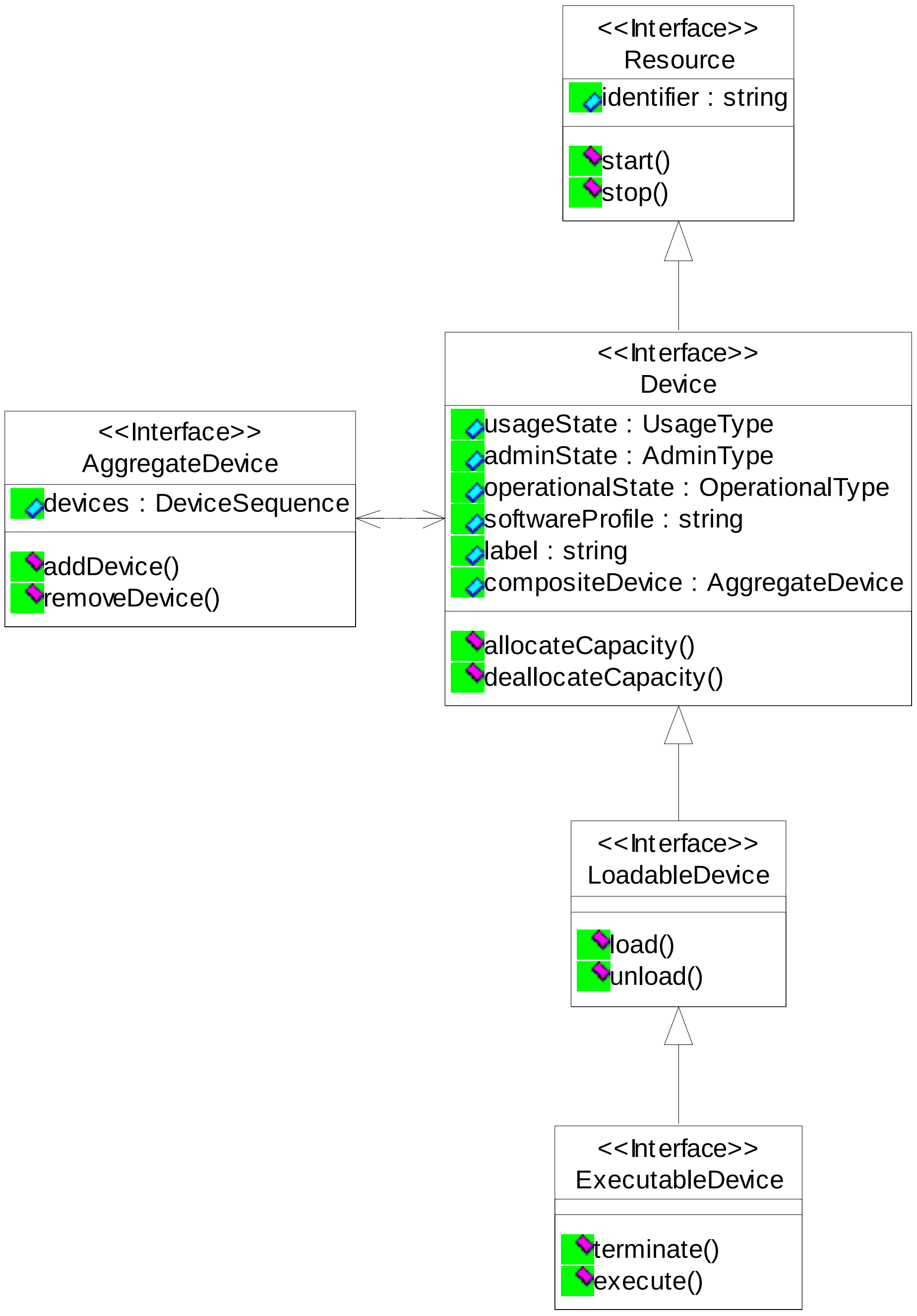


Figure 3-5. Logical Device Interface

3.3.1 OS Services.

No further explanation necessary.

3.3.2 CORBA Services.

No further explanation necessary.

3.3.3 CF Interfaces.

No further explanation necessary.**Profile.**

No further explanation necessary.

3.4 GENERAL SOFTWARE RULES.

3.4.1 Software Development Languages.

3.4.1.1 New Software.

The goal of new JTRS application development is to produce SW that is abstracted from the details of the HW platform and Operating Environment. Abstracting the details of the HW reduces/eliminates portability issues when changing processors, memory maps, bus protocols, etc. This does not guarantee 100% portable applications but it does greatly reduce the cost in porting SW to new platforms.

There are, however, applications that require performance that lower level languages provide. The use of lower level languages is allowed for special purpose functionality having requirements that cannot be met by a High Order Language. Use of Low Level languages should be minimized to reduce their impact on portability and reuse. Interfaces to functionality provided by Low level languages should be as abstract as possible to reduce the scope of the “less portable” SW functions.

3.4.1.2 Legacy Software.

It is recognized that the use of legacy SW can have significant cost and or schedule benefits for some applications. When legacy SW is to be ported to a JTRS architecture, a Higher Order Language is not mandatory. It is necessary, however, that the legacy SW be “wrapped” within a SW interface that is compatible with the Core Framework. This facilitatesdeployment of Legacy resources into the system and provides interface connectivity through CORBA. The Legacy application thus appears to the outside world as a standard JTRS resource, while retaining the Legacy implementation. This helps greatly to reduce the impact of porting Legacy SW.

4 HARDWARE ARCHITECTURE DEFINITION.

No further explanation required.

4.1 BASIC APPROACH.

See section 4.2

4.2 CLASS STRUCTURE.

Attributes

Attributes can be viewed from two viewpoints within the hardware domain:

1. Generic HW module descriptive information.
2. Specific HW module descriptive information utilized by the radio Core Framework and/or software applications.

In the development of the hardware section of the SCAS, it was the finding of the Hardware Working Group (HW WG) that attributes shown in the SCAS could be approached only as representative lists. It was recognized that the true description of any hardware object / device / module is its documented specification. It was further recognized that any simple listing of module attributes could never fully describe a HW module's characteristics. Within the current framework of the HW architecture, the module specification continue to serve the purpose of fully documenting the mechanical, electrical, and behavioral characteristics.

Attribute lists shown in the HW class diagrams in Section 4 of the SCAS represent a starting point for module specifications and potential use in *Device* Profiles. The actual attribute set for any HW module specification is a function of specific radio requirements. The actual attributes reported to the Domain Manager through the *Device* Profiles is a function of the software applications requiring HW information for operation. It is expected that module specifications is essentially time-invariant during the lifetime of the hardware, but it is possible that the *Device* Profiles can be updated over the lifetime of the hardware as more waveforms and more sophisticated software applications are installed in the radios.

Class Structure

As a counterpart of the graphical representation of software, hardware is also depicted in the Object-Oriented UML format. When Object Orientation is looked at closely from the hardware view, it can be seen that it essentially describes the way that hardware has always been approached (i.e., partitioned into circuit cards and modules), but maybe not optimally.

The hardware class structure portrayed in the SCAS is provided to the radio designer as a template for the radio design process. It breaks the radio into small building blocks that are functional in nature. These functional building blocks are the hardware classes and subclasses. Figure 4-1 shows a simplified design process that the radio designer might use in designing a JTRS radio. Using this approach as a radio is being designed, functional/performance allocation is first performed on the functional entities without regard to the physical nature of the ultimate hardware modules. As these allocations are performed, the attributes of the classes are assigned values. This mitigates premature association of function or performance to any particular hardware module. Once these functional and performance allocations have been fully performed, the radio designer can then proceed to the physical partitioning step to determine which functions coexists within any given physical module. This facilitates optimal functional partitioning to physical modules based on the physical/ mechanical/ environmental requirements of the radio as compared to available hardware circuitry. This implies size/cost/performance/approach trades including the potential utilization of COTS/GOTS hardware modules.

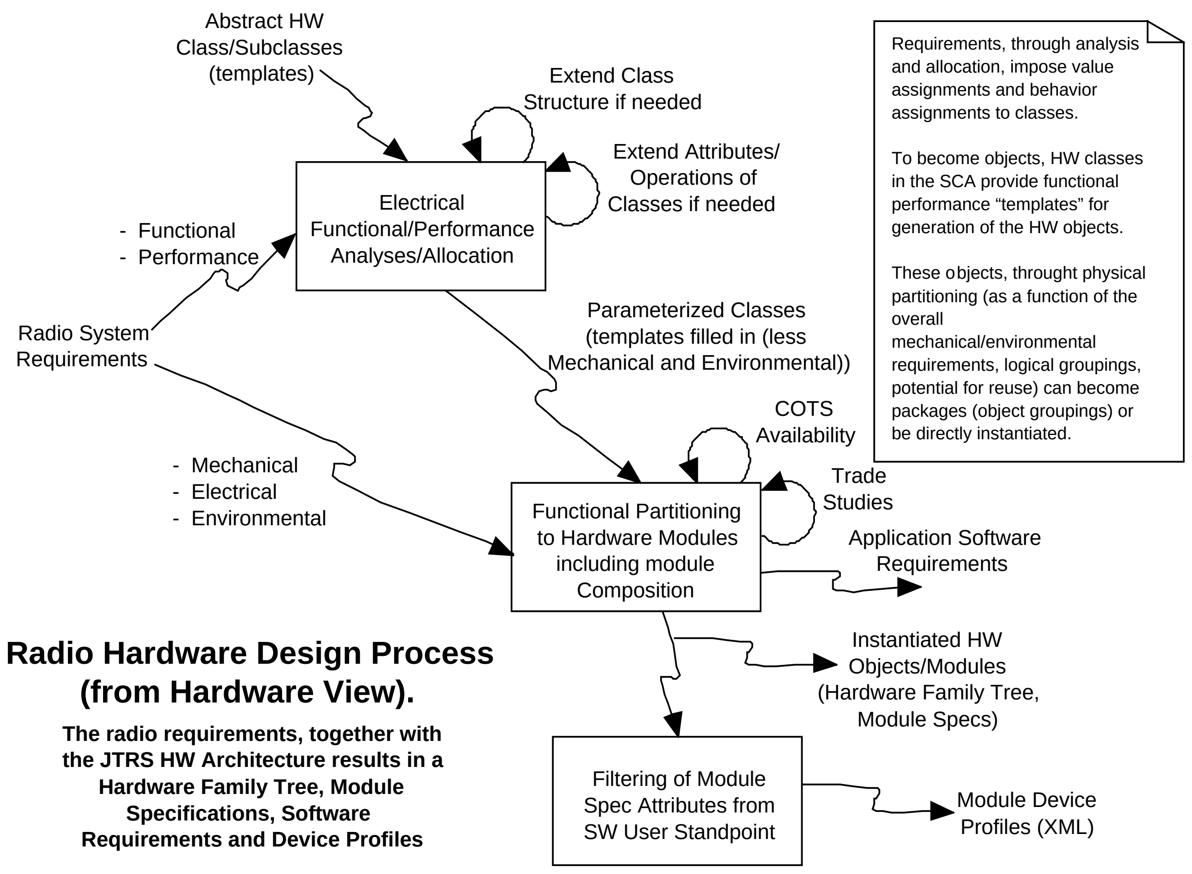

Abstract HW Class/Subclasses (templates)
Extend Class Structure if needed
Extend Attributes/ Operations of Classes if needed
Electrical Functional/Performance Analyses/Allocation
- Functional
- Performance
Radio System Requirements
Parameterized Classes (templates filled in (less Mechanical and Environmental))
COTS Availability
Trade Studies
- Mechanical
- Electrical
- Environmental
Functional Partitioning to Hardware Modules including module Composition
Application Software Requirements
Instantiated HW Objects/Modules (Hardware Family Tree, Module Specs)
Filtering of Module Spec Attributes from SW User Standpoint
Module Device Profiles (XML)
Radio Hardware Design Process (from Hardware View).
The radio requirements, together with the JTRS HW Architecture results in a Hardware Family Tree, Module Specifications, Software Requirements and Device Profiles
Requirements, through analysis and allocation, impose value assignments and behavior assignments to classes.
To become objects, HW classes in the SCA provide functional performance "templates" for generation of the HW objects.
These objects, throught physical partitioning (as a function of the overall mechanical/environmental requirements, logical groupings, potential for reuse) can become packages (object groupings) or be directly instantiated.

Figure 4-1. Radio Hardware Design Process

4.2.1 Top Level Class Structure

Stereotypes were utilized to generalize (or abstract) attributes in much the same way that higher level class structures generalize lower level classes. This was considered to be a key concept in understanding the role and relevance of attributes in the Object Oriented approach to hardware. In the HW class structure of the SCAS, potential software users, or "customers", of the attributes use stereotypes to sort or classify attributes. The abstraction of attributes into logical groups helps with the understanding and efficient tracking of attributes.

Also see section 4.2.

4.2.2 *HWModule(s)* Class Structure.

Programmability

In SCAS draft versions 0.2 through 0.4, the concept of programmability was addressed by creating programmable and non-programmable classes. The programmable class represented hardware that had no functionality other than that provided by software running on the programmable hardware that was loaded by system software. The non-programmable class represented hardware that, due to its circuitry (i.e. dedicated hardware devices), had embedded functionality.

The shortcoming of this approach was quickly realized when it became apparent that all (or most) basic class concepts produce programmable and non-programmable versions of the class. In order to stop the arbitrary proliferation of classes, the concept of programmability was captured as an attribute set or stereotype of the top level HW Module(s) class. All functional module classes could be realized as programmable, or non-programmable, as required. Multiple hardware functionality composed in a single hardware module can have the "programmable" attribute of each function, or sub-function, set as either programmable, or non-programmable, without the need for creating a new class.

The rationale, employed in developing the present approach for programmability also addresses the inherent programmable nature of a hardware object. Module developers can make the programmable nature of a particular hardware object available for use by the system software by proclaiming it with the "programmable" attribute. Hardware that is programmable obtains all or part of its functionality from software loaded onto it. On the other hand, a developer may develop an embedded function. A function realized by a software application running in a processor, is not provided from outside the module through system interfaces. For this latter case the programmability attribute may be set as non-programmable. In this case, the hardware class is not programmable, and its functionality is that of the embedded software and/or hardware.

Extensibility

The class structure provides a structured manner of grouping similar kinds of hardware together, recognizing that similar types of hardware generally have similar kinds of attributes. In this manner, like-hardware device properties can be compared and managed. In a similar manner, stereotypes are included to group attributes according to usage by user applications.

In developing the hardware architecture, it was realized that the notions of extensibility / expandability / extendibility must be an inherent part of the architecture. The architecture must provide freedom for new developments, while not placing a bound on creativity.

The hardware class structure in the SCAS is considered representative of current radio systems, but it is not meant to be all-inclusive for all implementations, nor to limit the description of hardware that does not fit into this structure. The addition of new technology may be represented by additional hardware classes. Also, information not adequately represented may be described by the addition of new or different attributes to existing classes. Attributes associated with applications not presently described may be grouped by the addition of new stereotypes.

GPS

The requirement of JTRS support for the GPS waveform is clearly identified in paragraphs 4.a.(2).(f) and 4.a.(2).(n) of the ORD for JTRS. For consistency with the OO approach to hardware and the implementation of waveforms, it was thought that GPS should be treated as any other waveform, with the exception that it is required for all domains and all platforms. However, early in the development of the hardware section of the SCAS, it was apparent that the hardware implementation for the GPS waveform using a mix of hardware devices, as for other waveforms, was not currently cost effective.

With the advance of technology, GPS receivers in the military and commercial markets have progressed to the point that COTS/GOTS hardware is extremely compact and relatively inexpensive. Given the cost advantage of COTS/GOTS hardware and the JTRS architecture's ability to accommodate legacy hardware, it was decided to represent GPS as a separate and distinct hardware module class (see Figure 4-2 in the SCAS). However, if compelling reasons point to the implementation of the GPS waveform using a mix of hardware devices, rather than an embedded and dedicated legacy receiver, the SCAS provides the flexibility for such an approach.

4.2.3 Class Structure with Extensions.

No further explanation required.

4.2.3.1 *RF* Class Extension.

No further explanation required.

4.2.3.2 *Modem* Class Extension.

No further explanation required.

4.2.3.3 *Processor* Class Extension.

No further explanation required.

4.2.3.4 *INFOSEC* Class.

No further explanation required.

4.2.3.5 *I/O* Class Extension.

No further explanation required.

4.2.4 Attribute Composition.

No further explanation required.

4.3 DOMAIN CRITERIA.

No further explanation required.

4.4 PERFORMANCE RELATED ISSUES.

The architecture must allow for the arbitration and mitigation of potential interference pertaining to the use of the radio frequency spectrum for each included channel. Not only should the radio system address interactions between assigned channels it should be flexible enough to account for, and be tailored to, platform conditions. The architecture needs to address this coupling of real world installed environmental conditions into radio system performance. To accomplish this, the architecture should support incorporation of dedicated cosite mitigation hardware either self contained in the radio chassis, or externally within an ancillary unit that are referred to as an Antenna Distribution Unit (ADU). Additionally the architecture must allow for the incorporation of a software cosite manager function that can utilize knowledge of the radio (hardware), waveform (software) and platform (installation) parameters to perform real time predictions of potential cosite interactions.

These goals are accomplished within the hardware architecture through the utilization of the flexibility afforded with the JTRS OO approach. Specifically, within the hardware architecture, cosite has been addressed at both the class/subclass and attribute level. Figure 4-2 shows representative subclass and attribute relationships. The RF, Modem and I/O classes, subclasses

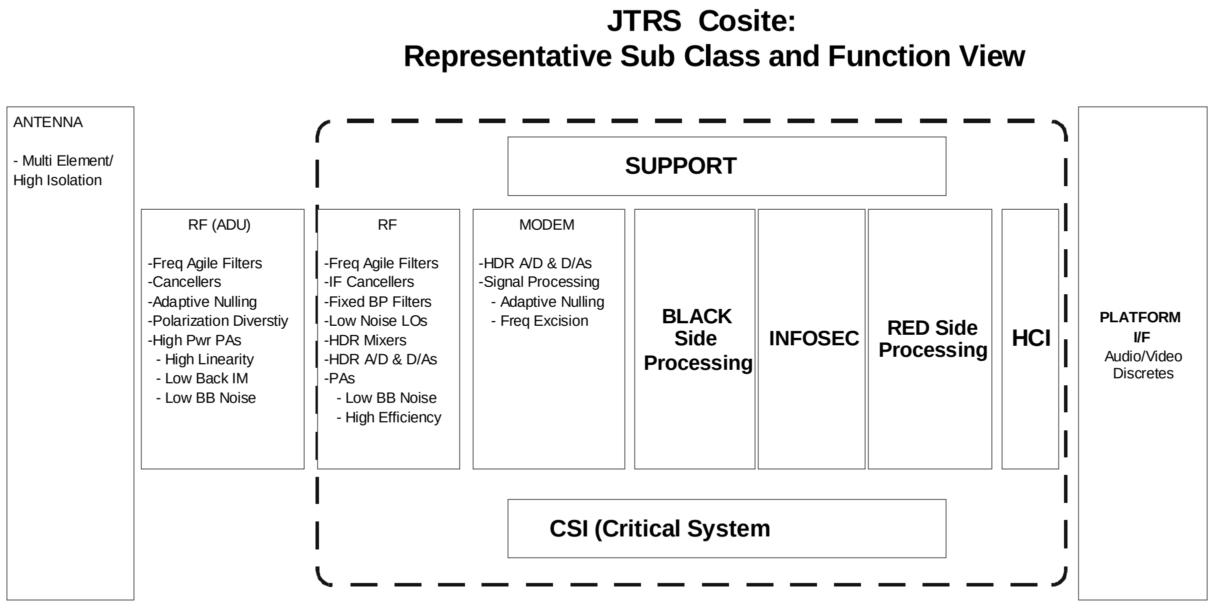


Figure 4-2. Cosite Subclasses and Attributes

and attributes support the definition and substantiation of the cosite mitigation hardware and its associated performance. To further help define the cosite interactions between classes a cosite stereotype <<CositePerformance>> has been created in the RF top class and is inherited by all RF subclasses or class extensions. Additional detail for the cosite stereotype has been added to the Receiver, Exciter and Power Amplifier subclasses (see SCAS, Figure 4-3, RF Class Extensions). Finally, recognizing the strong dependency of dedicated cosite mitigation devices to the RF arena, a separate cosite subclass was added to the RF class. This object-oriented approach enables a consistent application of the HW architecture (classes and rules) across the various domains (i.e., Handheld, Dismounted, Vehicular, Airborne, and Maritime/Fixed) and allow for a JTRS installation into platforms with anything from a mild to severe cosite environment. Examples of how cosite solutions can be realized within the JTRS framework are provided below.

Simultaneous contention for frequency spectrum by multiple transmitters and receivers must be eliminated. Degradation is usually inversely related to frequency separation, and may be mitigated by several methods that are supported by the SCAS.

The SCAS supports specialized hardware that is often employed to decrease cosite related degradation. This is supported via an RF subclass for cosite mitigation hardware. SCAS also supports software techniques, such as cosite manager, via attributes useful to such an application. A cosite manager application can authorize emissions or command protective measures based on instantaneous conditions. Protective measures, (for example inserting attenuation, blanking a signal, or overriding AGC), while usually degrading, are usually preferred to uncontrolled conflicts. Other protective measures, such as time slot scheduling, may not degrade operation.

Examples of cosite mitigation hardware are preselectors, postselectors, cross-band filters, multicouplers, and signal canceller devices. The performance of all such devices can be characterized by the properties of the attenuation and bandwidth attributes. For example, the system performance of signal cancellation devices can be described using these attributes to report cancellation depth (attenuation) and bandwidth. Similarly, filter characteristics are defined by values reported by these attributes. Additional information, such as tuning speed, and frequency, relevant to cosite mitigation devices is already available as attributes "inherited" from the general RF class.

A Cosite Manager application is supported by the SCAS to provide software control of assets to mitigate frequency contention. The Cosite Manager must first recognize potential conflicts. In order to determine this, attributes (identified by the <<CositePerformance>> stereotype) provide source information / data. Examples of cosite performance related attributes are transmit wideband noise and receiver (or LNA) dynamic range (likely reported via a third order intercept or similar method).

Note that the users of attributes are not mutually exclusive. The cosite manager use other attributes, such as <<performance>> stereotyped attributes. Examples of these are PowerLevel, NoiseFigure, etc. Some cosite performance attributes such as dynamic range are also performance related. Information and data of software performance (for example DSP) and platform performance (for example antenna coupling) are needed by the cosite application in order to compute potential degradation and command mitigation.

4.5 GENERAL HARDWARE RULES.

No further explanation required.

4.5.1 Device Profile.

The use of attributes to describe hardware is the heart of the object-oriented approach of the SCAS. The entire hardware class structure is based on grouping these attributes according to device types (classes) and on methods of organizing them (stereotypes) according to application users.

The *Device* Profile is used to collect and report needed attributes and their values to the system software. The *Device* Profile for hardware devices consists of the Device Component Descriptor

File and the Device Descriptor File. The Device Profile and its files are detailed in SCAS 3.1.3.4. XML is the language chosen for the Device Profile, as described in SCAS section 1.3.1.3.

Attributes within the *Device* Profile are those attributes needed by software applications. The applications can be the Domain Manager or other applications such as waveform software. Some applications, such as a Cosite Manager (see section 4.4), may require more than single valued attributes, so these attributes may be presented as data files or links to data files.

4.5.2 Hardware Critical Interfaces.

Early in the hardware structure definition, the hardware working group recognized that critical hardware interfaces should be a function of the point of sale. In other words, if you are buying a radio, then the critical interfaces are the radio box inputs and outputs. If you are buying modules, then the critical interfaces are the module inputs and outputs. Recognizing that the effective point of sale for JTRS radios is both at the radio and module level, (since JTRS radios are required to be modular per the ORD), the definition of a critical interface must take the more detailed view (module level). Thus, the SCAS defines all inputs and outputs from hardware modules to be critical interfaces. This provides the benefits of allowing competitive procurement of HW modules as well as making HW modules available for reuse in other radios.

Recognizing that defining critical interfaces at this level provided benefits, it was further determined that there was no additional benefit from requiring interfaces to be defined in any more detail (i.e., internal interfaces within a module). Defining critical interfaces at the module boundaries and not inside module boundaries helps to protect hardware developers from having to share proprietary rights/intellectual property in order to sell radios and/or modules. In other words, one company could develop a module with certain (critical) interfaces and performance attributes using discrete circuitry. Later, another company could supply a module to the same exact specification (external interfaces and performance) using a proprietary integrated circuit. The interfaces internal to the module in either form would not need to be revealed in detail. This is not meant to prevent a supplier from making internal interfaces available at module boundaries either for other potential radio uses or for proprietary programming purposes.

Interfaces at module boundaries are required to be in accordance with commercial or government standards to foster use of COTS/GOTS hardware and to foster reuse of newly developed modules. This should also make it easier and more attractive for commercial hardware suppliers to become hardware suppliers to military radios.

Unique interfaces are permissible when warranted by radio performance requirements. This might include overload of the selected standard bus structure or other specific requirements that a radio must meet. Requiring these exceptional interfaces to be clearly and openly defined maintains the benefits of competitive procurement and reuse while eliminating technological roadblocks.

4.5.2.1 Interface Definition.

See SCAS section 4.5.2.

4.5.2.2 Interface Standards.

See SCAS section 4.5.2.

4.5.2.2.1 Interface Selection.

See SCAS section 4.5.2.

4.5.3 Form Factor.

Commercial standards provide the maximum ability to re-use existing developments, provide technology insertion, and leverage COTS technology growth as much as possible. This is not possible with proprietary standards that impede open system architectures. At the implementation level (below the architecture level of the SCAS), defining and encouraging specific form factors provides the maximum interchangeability and reuse of modules. However, no single form factor is applicable across all domains. Further, no form factor can be totally standardized within a given implementation. Certain unique items like power supplies and reference standards may not be conducive or may not fit in a chosen standard. The ORD recognizes this when it states that the JTRS system "shall provide an internal growth capability through an open systems architecture approach … and is modular, scaleable, and flexible in form factor[3]." Using a "should" rather than a "shall" in SCAS section 4.5.3 is due to the need to maintain flexibility in the form factor.

4.5.4 Modularity.

In tracking compliance with the Joint Technical Architecture, the concept of modularity has been established as a requirement in the ORD for the JTRS. Though the concept of modularity is common in the development and construction of today's electronic systems, modularity has been directly addressed as a threshold Key Performance Parameter (KPP) requirement in paragraph 4.a.(1).(b)) of the ORD. In doing so, the ORD has emphasized scalability and flexibility in the form factor, which has directly influenced hardware rules in the SCAS.

[3] Operational Requirements Document for Joint Tactical Radio, 23 March, 1998, (1) General Performance Requirements (a)

5 SECURITY ARCHITECTURE DEFINITION.

Security for the JTRS radio not only encompasses the radio implementation but also the architecture as well. Traditionally, security analysis has been relegated to the physical domain. This section of the SRD discusses rationale for various security requirements that have been included in the SCAS and its companion document, the Security Supplement. The architectural requirements listed are an abstraction of requirement from the UNIFIED INFOSEC CRITERIA, UIC; which contains both architectural and implementation details.

The requirements are organized into three major categories: cryptographic (Crypto), non-cryptographic (NC) and system. Requirements overlapping categories were placed in a 'best fit' category.

The cryptographic category includes those architectural requirements that affect the COMSEC subsystem. This category includes specific cryptographic functions and the support required to execute those functions. The non-cryptographic category includes those architectural requirements that are within the INFOSEC boundary. The INFOSEC boundary includes the COMSEC subsystem and the controls that directly affect the COMSEC subsystem. The system category includes those architectural requirements that have security implications from a radio perspective. This category includes HCI, BIT, power, etc.

The requirements discussion includes a requirements description, whether the requirement is architecture or implementation, where it fits in the architecture and what implied requirements exist.

Of the three major categories, the user application functions within the JTRS radio have been further divided into six categories as follows:

- Crypto Functions
- Crypto Support Functions
- NC Security Functions
- NC Security Support
- System Application Functions
- System Functions

These six categories are addressed, along with their subcategories, in terms of the following issues.

- Is it architecture and/or implementation?
- Where does it fit within the JTRS architecture?
- What its implied requirements are?
- What is the description of the category/subcategory?

In some cases, the formulation of the details is still in process but the functions, without detail, are listed for completeness.

5.1 CRYPTO FUNCTIONS.

5.1.1 Encrypt/Decrypt.

Description: Encryption is a means of enciphering and encoding data for secure transmission. Decryption is the reverse roll of encryption, were the data is decoded and deciphered.

Architecture/Implementation: The architecture allocates this function to the Cryptographic Subsystem (CS/S). However, the actual encryption/decryption process is an implementation issue based on the Crypto algorithm being performed.

Place in the Architecture: CS/S.

Implied Design:

- Data for encryption is isolated from Bypass data
- The actual encryption is a Implementation that is performed by the CS/S.

5.1.2 Crypto Alarm.

Description: Equipment Alarms involve monitoring specific components of the system for evidence of failures that could result in security problems. The monitoring of these alarms and their indicators must be designed into the system.

Architecture/Implementation: Alarm functions become part of the Architecture within the CS/S and are also part of the implementation of the system. The Alarm indicators become part of the architecture along with the action taken when an Alarm occurs. The actual communication between modules within the system is part of the system implementation.

Place in the Architecture: CS/S and HCI

Implied Design:

- Report Alarm condition to the operator.
- Provide an alarm indicator on the front panel of the terminal.
- Shut down all cryptographic function until alarm condition is resolved.
- The implementation must have the hooks for data transmittal of the alarm condition throughout the system.
- Supports isolation of an Alarm to a channel when appropriate.
- Inhibits all cipher text output on channel in Alarm
- Shut down all channels if a catastrophic Alarm occurs

5.1.3 Zeroize All.

Description: Removal or elimination of keys from a Crypto equipment or storage.

Architecture/Implementation: Architecture, The architecture must be capable of stimulation of the Zeroize command for both internal and external commands. Zeroization of keys can be performed in several manners, which effect the architecture.

Place in the Architecture: CS/S and HCI

Implied Design:

- Activation of Zeroize process by an active signal.
- Zeroization of Keys due to a TAMPER detect signal going active.
- Zeroization of Keys due to a non-scheduled power loss.

5.1.4 Selective Zeroization.

Description: Removal or elimination of a key or keys from a Crypto equipment or storage.

Architecture/Implementation: Architecture, The architecture must be capable of stimulation of the Zeroize command for both internal and external commands. Zeroization of keys can be performed in several manners, which effect the architecture.

Place in the Architecture: CS/S and HCI

Implied Design:

- Activation of Zeroize process by an active signal.
- Selective zeroization of keys
- Activation of zeroization process by an Over The Air Zero (OTAZ) message.

5.1.5 Programmable Crypto.

Description: A set of hardware and software capable of changing COMSEC algorithms and keys (Need unwrap key) in a tactical environment. This allows the device to be interoperable with several different COMSEC devices.

Architecture/Implementation: Architecture pertaining to the CS/S, The architecture needs to supply the I/O required for interfacing to the different possible programmable Crypto devices.

Place in the Architecture: CS/S

Implied Design:

- Provide secure storage of Crypto algorithms.
- Provide secure key storage.
- Provide interoperability with legacy waveforms.
- Must conform to all FSRS requirements which legacy Crypto fulfilled.
- Association of algorithm and keys to instantiated channel.

5.1.6 External Cryptography Support.

Description: A piece of cryptography, which is external to the system. This external cryptography communicates with the system via I/O ports.

Architecture/Implementation: Architecture, The architecture must support the communication path to the external cryptography.

Place in the Architecture: HCI, CS/S

Implied Design:

5.1.7 Bypass Control, Data, and Protocol.

Description: Cryptographic bypass is the function of routing information around the Crypto logic. The bypassed data could be control/status information, Plain text bypass of data, or protocol information.

Architecture/Implementation: Architecture, In the area of Plain text bypass a deliberate action by the operator must take place before plain text data can be bypassed. The bypass of control/status and protocol information falls on both the architecture and implementation. The architecture must provide a means of routing the information for bypass.

Place in the Architecture: CS/S

Implied Design:

- Provide a Bypass for communications protocol/Header based on information content.
- Provide a Bypass for control/status information not associated with received/transmitted data.
- Provide a local access control mechanism for Bypass
- Monitor Bypass channel

5.1.8 Security Policies.

Description: Security policies govern the behavior of the Control/Status and Header bypass mechanisms. They may be changed without affecting the mechanisms themselves. Multiple policies are in effect simultaneously.

Architecture/Implementation: Policy format and instantiation (Architecture). Mechanism of enforcement (Implementation).

Place in the Architecture: CS/S

Implied Design:

- Identification
- Authentication
- Integrity

5.1.9 TRANSEC.

Description: TRANSEC is a measure used to protect transmissions from interception and exploitation.

Architecture/Implementation: Both Architecture and Implementation. The architecture needs to be able to handle where the TRANSEC operation are performed (i.e., RED side/ BLACK side). The actual performance of the TRANSEC function is an implementation issue when being performed either on the CS/S or on the Modem.

Place in the Architecture: CS/S, BLACK side (Processor, Modem)

Implied Design:

- Support the execution of TRANSEC (i.e. Key load algorithm)
- Load time
- Maintain time (with or without power)
- Use TOD

5.1.10 Multi-Channel Operation.

Description: Be able to instantiate and operate multiple waveform channels and therefore multiple crypto channels simultaneously.

Architecture/Implementation: Architecture pertaining to the CS/S and for MILS, the Red side.

Place in the Architecture: HCI, CS/S, Red Side.

Implied Design:

System High Operation

- The CS/S inhibits instantiation of channels of different security classification levels for red-black encryption and black-red decryption.
- The CS/S signals the Red side to zeroize its memory if the classification level of operation is changed to a lower classification level.

MILS Operation

- The CS/S inhibits information labeled at one security level from being processed by a cryptographic channel instantiated at a different security level.
- The CS/S signals a Red GPP to zeroize memory used to process data at one classification level prior to reusing it to process data at a lower classification level.

5.2 CRYPTO SUPPORT FUNCTIONS.

5.2.1 Digital Signature (High Grade).

Description: Accept Block message (s) for signature validation.

Architecture/Implementation: Architecture pertaining to the CS/S.

Place in the Architecture: CS/S.

Implied Design:

- Is capable to select what kind of signature.
- Select PKI certification or Key.
- Submit file length.
- Result returned (meaning if digital signature was unwrapped).

5.2.2 External Algorithm load.

Description: Download of Crypto Algorithm from an external means.

Architecture/Implementation: Architecture in the support of DS101, user interfaces.

Place in the Architecture: HCI, CS/S.

Implied Design:

- Selection of the Algorithm.
- Associate the algorithm ID with the proper algorithm. (Binding, Tagging)
- Authentication.
- Integrity check.

5.2.3 Internal Algorithm load.

Description: Loading of internally stored algorithms into the Crypto by ID.

Architecture/Implementation: Implementation.

Place in the Architecture: CS/S.

Implied Design:

- Decryption
- Authentication
- Integrity
- Identification
- Verification and test

5.2.4 BLACK External Load Key.

Description: Key load is the method of loading key either from a key fill device or over the air.

Architecture/Implementation: Implementation.

Place in the Architecture: CS/S and HCI

Implied Design:

- Provide fill operations for a minimum of BLACK fill up to Benign fill capability.
- Encrypt store
- Decrypt check
- Integrity

5.2.5 RED External Load Key.

Description: Key load is the method of loading Key from either Key fill device (i.e. DTD, KYK-13, KOK-13, KOI-18…) or by an Over The Air Rekey Message.

Architecture/Implementation: Architecture, Keys are loaded via two methods which are, direct fill and by an OTAR message. The direct fill are done via a common fill port. The fill lines from the fill port go directly to the CS/S. The CS/S steer the keys directly to the key storage location (i.e. RAM Storage or internal storage to the COMSEC device).

Place in the Architecture: CS/S and HCI

Implied Design:

- Provide fill operations for a DS101/DS102 interface
- Provide capability for DS100 key tagging
- Provide capability for Multi key for different Algorithm
- Provide for OTAR capability for supporting waveforms
- Encrypt and store

5.2.6 Use of Key.

Description: The use of the key in performing the COMSEC and TRANSEC functions of the waveform.

Architecture/Implementation: Architecture, Interfaces requirement with the operator.

Place in the Architecture: HCI

Implied Design:

- Unique key ID’s associated with each Key
- Validation of key
- Selected key
- Decrypt
- Lead outside Boundary (e.g. TRANSEC key)

5.2.7 External Zeroize.

Description: The zeroization of key from an external means.

Architecture/Implementation: Architecture, The architecture must provide the means for handling an input from a user interface.

Place in the Architecture: HCI, CS/S

Implied Design:

- Provide means to respond to an external zero-all switch.
- Means of zeroization by the operator using a mode Zeroize switch.
- Means of zeroization by means of a Tamper detect and under/over voltage.

5.2.8 Synchronize/Resynchronize.

Description: Algorithm dependent, bit alignment/Frame Alignment for Crypto sync.

Architecture/Implementation: Both Architecture and Implementation. The architecture needs to be able to handle where the Sync/Resync operation are performed (i.e. RED side/BLACK side). The actual performance of the Sync/Resync is an implementation issue.

Place in the Architecture: CS/S, BLACK side (Processor, Modem).

Implied Design:

- Support execution of Sync/Resync.
- Time base
- Get timing
- Use Randomizer (Message Indicator based)

5.2.9 Sense Patterns.

Description: These are message patterns contained within the message.

Architecture/Implementation: Architecture, The Architecture needs to handle real time messaging from user I/O control or radio I/O control.

Place in the Architecture: RED/BLACK Side, HCI

Implied Design:

- Waveform dependent detective device which notifies the user
- Control performance of automated function (SICGARS, EOM, and OTAT…)

5.2.10 Time Of Day (TOD).

Description: Time tracking for cryptographic sync.

Architecture/Implementation: Both architecture and implementation. Architecture needs to carry the real time (e.g. GPS, Net controller) clock function. The actual communication of the TOD between modules within the system is part of the implementation.

Place in the Architecture: Modem, CS/S, RED/BLACK side of terminal.

Implied Design:

- Battery to real time clock
- Potential use of control Bypass
- Get time
- Provide or validate time stamp

5.2.11 Randomization.

Description: Generation of random numbers for multiple purposes (i.e. CIK, Crypto randomization Patterns).

Architecture/Implementation: Implementation

Place in the Architecture: HCI, CS/S

Implied Design:

- User actions for timing

5.2.12 Crypto Control.

Description: Control of Crypto functions from General Purpose Processors (GPPs).

Architecture/Implementation: Architecture and Implementation. External interfaces (APIs) defined for accessing the Crypto functions (Architecture). Translation of external interfaces to internal Crypto functions (Implementation).

Place in the Architecture: CS/S, Security APIs

Implied Design:

- Controlled Access to Crypto functions

5.2.13 Over The Air Rekey (OTAR).

Description: A method of rekeying a user channel over the air.

Architecture/Implementation: Both Architecture and Implementation. Architecture must provide the means to detect the OTAR, either by the CS/S or the Modem. The implementation is the passing of the information between modules.

Place in the Architecture: CS/S, Modem

Implied Design:

- Notify user of OTAR receipt
- Assign key ID for received OTAR
- Key update
- Wrap key for storage

5.2.14 Over The Air Zeroization (OTAZ).

Description: A method of zeroization of key or keys by means of an over-the-air message.

Architecture/Implementation: Architecture, The architecture must supply the means for the detection of the OTAZ message by either the CS/S or the Modem.

Place in the Architecture: CS/S, Modem

Implied Design:

- Pattern detection capability for the Modem and CS/S
- Authenticate command

5.2.15 Over The Air Transfer (OTAT).

Description: A key file transfer by means of over the air communication.

Architecture/Implementation: Implementation

Place in the Architecture: CS/S/key storage

Implied Design:

- Data storage
- User notification
- Message authentication/integrity
- KEK selection

5.2.16 HCI (Security Relevant Status Functional).

Description: The HCI is able to identify algorithm and key IDs contained within the system.

Architecture/Implementation: Both.

Place in the Architecture: HCI, Programmable Cryptography, Key Manager, and Domain Profile.

Implied Design:

- The system administrator is able to retrieve system configuration and status information.
- This information is needed to ascertain the system configuration information in preparation to mission planning.
- Security critical configuration information includes algorithm and key status.
- The algorithm and key status information uniquely identifies each algorithm and key within the system.
- The system administrator is then able to identify algorithms and keys that may need to be loaded into the JTRS prior to the next mission.

5.3 NON-CRYPTOGRAPHIC SECURITY FUNCTIONS.

5.3.1 Initialization.

Description: Generation of Crypto/sync for encryption/decryption.

Architecture/Implementation: Architecture, In receive mode the Modem must have the capability to detect and send to CS/S.

Place in the Architecture: Modem/CS/S

Implied Design:

- Detection of sync patterns

5.3.2 Digital Signature II (Low Grade).

Description: A digital signature algorithm is used by a signatory to generate a digital signature on Hashed data and by a verifier to verify the authenticity of the signature. An asymmetric algorithm is used for digital signatures.

Architecture/Implementation: Both. The architectural issues are related to message structure support. Implementation is the primary issue.

Place in the Architecture: Red processing/Black processing. Architecture does not prevent you from handling in INFOSEC boundary.

Implied Design:

- Digital signatures is used to provide integrity and authentication mechanisms.
- The architecture supports digital signature messages.

5.3.3 Authentication.

Description: To establish the validity of a claimed identity. The establishment of identity may be provided by different mechanisms depending on the strength of mechanism required. Authentication may be as simple as using user-name and password pairs if weak mechanisms are all that is required. Stronger mechanisms can include public key/private key cryptography or Crypto ignition keys (CIKs). Hardware tokens, using public key/private key cryptography and passwords, may be used for stronger authentication, e.g. MISSI Fortezza cards.

Architecture/Implementation: Both. The architectural issues are related to message structure support. Implementation is the primary issue.

Place in the Architecture: Within the INFOSEC Boundary..

Implied Design:

- The authentication mechanism resides within an INFOSEC boundary in the JTRS.
- Accommodate table privileges, token interfaces
- Return results and execute

5.3.4 Privilege establishment.

Description: Privileges refer to actions permitted by individual system entities.

Architecture/Implementation: The architecture supports the assignment of privileges for various entities in the JTRS. Implementation provides for the management of the privileges.

Place in the Architecture: The architecture provides the capability for the assignment of privileges to core applications, non-core applications, CORBA ORB entities, and I/O devices based on the security policy.

Implied Design:

- The architecture provides the capability for the assignment of privileges to various entities within the JTRS system. (Entities may be core applications, non-core applications, CORBA ORB entities, and I/O devices, system administrators, radio network users, or input/output ports.)

- The security policy identifies the privileges of each system entity.

5.3.5 Data Separation.

Description: Data separation is supported by the access control mechanism and access control database. The access control mechanism and access control database provide for secure message routing within the JTRS architecture.

Architecture/Implementation: Both. The architectural issues are in the existence of the access control mechanism and access control database. The implementation of these security functions is left to the developer.

Place in the Architecture: Access control mechanism and access control database.

Implied Design:

- No additional architectural impacts are required.

5.3.6 Application Separation.

Description: Application separation is supported by the separation mechanism. The separation mechanism provides for process separation between applications.

Architecture/Implementation: Both. The architectural issues are in the existence of the separation mechanism. The implementation of this security function is left to the developer.

Place in the Architecture: Separation mechanism.

Implied Design:

- No additional architectural impacts are required.

5.3.7 Major Function: Access Control.

Separation of data is a primary concept in computer security. In secure computers, this concept can take at least three forms: trusted computer base (TCB), access control mechanism, and separation mechanism. The TCB is the security-critical (trustworthy) part of the system; all other parts are presumed to be hostile (fraught with Trojan horses, for example).

The access control mechanism is a subpart of the TCB. The access control mechanism arbitrates access requests to the system resources and enforces those access decisions. The access decisions are based on a specified system security policy (often embodied in an access control list/database) and on the security attributes of the requestor and requested resource. See Figure 5-1 for a model of an access control mechanism implementation. The three design requirements that must be met by an access control mechanism are:

- The access control mechanism must be tamperproof;
- The access control mechanism must always be invoked;
- The access control mechanism must be small enough to be subject to analysis and tests, the completeness of which can be assured.

Figure 5-1 illustrates the implementation of an access control mechanism. The access-control database is the element of the system that embodies the security policy. The access control mechanism can be used to support different system security policies with changes to the access control database.

A separation mechanism is the subpart of the access control mechanism, which isolates system resources (memory, input/output devices, processing time, storage devices like disk, etc.) between various processes on the system. A common mechanism for implementing a separation mechanism is a memory management unit (with associated software). Thus, the access control mechanism provides for controlled sharing amongst the isolated domains created by the separation mechanism.

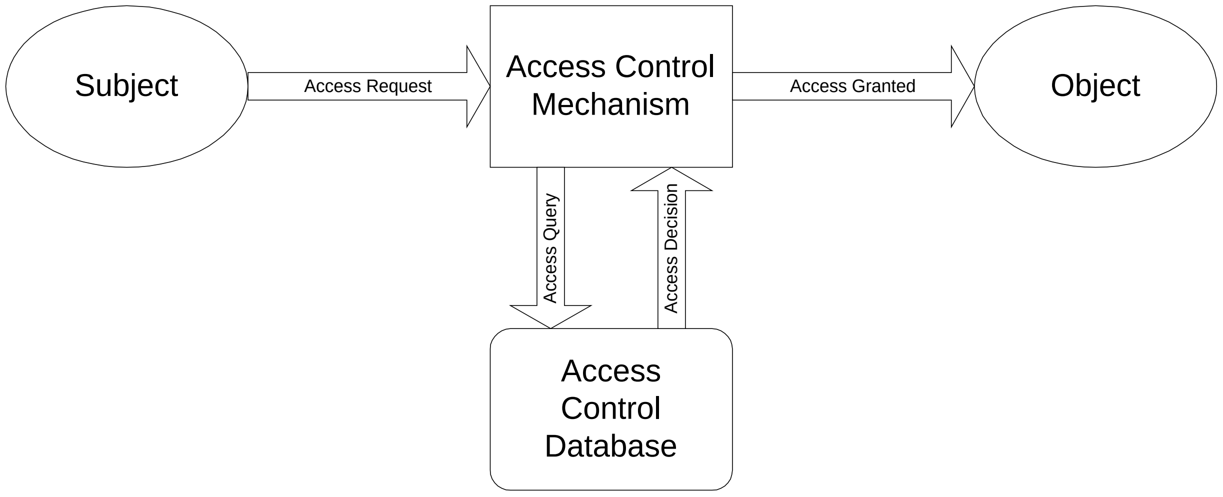


Figure 5-1. Generic Access Control Mechanism Model

5.3.7.1 <u>Sub-Function:</u> Access Control Mechanism.

<u>Description:</u> The access control mechanism arbitrates access requests to the system resources and enforces those access decisions. The access decisions are based on a specified system security policy (often embodied in an access control list/database) and on the security attributes of the requestor and requested resource.

<u>Architecture/Implementation</u>: Both. Access control mechanism applies to the architecture in providing a framework for securely establishing non-core applications, and secure messaging within the architecture. The implementation develop the appropriate mechanism(s) instantiating the access control mechanisms.

<u>Place in the Architecture</u>: Core framework, Message Registration, Control/Status Bypass, Protocol Bypass, and User (system administrator and potentially other users) access.

<u>Implied Design:</u>

- Core framework implements the access control mechanism portion of the reference monitor concept for system control functions.

5.3.7.2 Sub-Function: Separation Mechanism (Access Control).

Description: A separation mechanism is the subpart of the access control mechanism, which isolates system resources (memory, input/output devices, processing time, software processes, mass storage devices, etc.) between various processes on the system.

Architecture/Implementation: Both. The existence of a separation mechanism is architectural. The implementation of the separation mechanism is left to the implementor.

Place in the Architecture: A unique separation mechanism exists on all Red-side nodes and Black-side nodes as applicable (waveform dependent).

Implied Design:

- The architecture supports a separation mechanism.
- Paths
- Processes
- Data

5.3.8 Audit.

Description: Audit provides a mechanism for recording significant events. The audit log is protected from accidental or malicious deletion or from being read by unauthorized entities. Audit information is provided to the system administrator and Security officer.

Architecture/Implementation: Both

The existence of audit is an architectural feature.

Place in the Architecture: Audit functionality for JTRS is provided by the operating environment.

Implied Design:

- Determine audible events
- Each entity designated as having audit responsibilities reports any auditable events to the audit mechanism.
- Protects the audit information from being tampered with or accessed by unauthorized entities.

5.3.9 Examples of Implementation (Firewall).

Description: The purpose of a firewall is to provide controlled and audited access to services, both from inside and outside an organization’s network, by allowing, denying, and/or redirecting the flow of data through the firewall. Firewalls fall into two major categories: traffic-filter and application-level firewalls.

Architecture/Implementation: Implementation: Firewalls may be used to support network management applications or the implementation of user interfaces.

Place in the Architecture: Not applicable.

Implied Design: None.

5.3.10 Classified Application.

Description: Classified applications refer to applications whose object code is classified.

Architecture/Implementation: The architecture is able to decrypt classified applications and securely route the classified object code to the appropriate process space.

Place in the Architecture: Security requirements for securely routing of classified object code is supported by the separation mechanism, access control mechanism, core framework, and secure message connectivity.

Implied Design:

- All classified object code is stored encrypted when not in use.
- The architecture supports process isolation for classified processes.
- The architecture supports message isolation for file transfer of classified applications.
- Erase Red application on certain applications (e.g., Declassified terminal)

5.3.11 Data Integrity.

5.3.11.1 Sub-Function: Internal Data Integrity.

Description: The state that exists when computerized data has not been exposed to accidental or malicious alteration. Internal data integrity provides assurance that messages within the JTRS have not been altered between client and server.

Architecture/Implementation: Both. The architecture supports data integrity services. The integrity service implementation is left to the developer.

Place in the Architecture: Integrity mechanisms supports message traffic within the architecture.

Implied Design:

- The operating environment supports data integrity mechanism for message traffic within the JTRS architecture.
- Integrity for programs applications check (e.g., files, tables)

5.3.11.2 Sub-Function: External Data Integrity.

Description: The state that exists when computerized data has not been exposed to accidental or malicious alteration. External data integrity provides assurance that messages received from outside the JTRS have not been altered from the trusted source. Data integrity can be implemented in conjunction with authentication services to provide secure distribution of data or software files.

Architecture/Implementation: Both. The architecture supports MISSI integrity services. The implementation may use Type I or Type II integrity services. Data integrity services may be implemented in hardware, software, or tokens. Cryptographic algorithm integrity uses an approved authentication service.

Place in the Architecture: Any integrity mechanism is accessible by the core framework. Cryptographic algorithm integrity is implemented within equipment authorized by the appropriate certification authority.

Implied Design:

- The operating environment supports integrity services (e.g., digital signal secure hash).

5.4 NON-CRYPTO SECURITY SUPPORT FUNCTIONS.

5.4.1 Security Policy.

Description: Security policy provides for the secure operation of the JTRS.

Architecture/Implementation: Both. Architecturally, the security policy is embodied in the access control database. The policy vary from program to program because of the different security policies of each program.

Place in the Architecture: Architecturally, the security policy is embodied in the access control database. Implementation of the access control database is left to the developer.

Implied Design:

- The security policy is embodied by an access-control database, which encapsulates the security policy in a way that system entities (e.g. access control mechanisms) may enforce the security policy.
- Security policies that are enforced at the system level is embodied in the Security policies.
- Security policies may also be enforced at a more local level. For example, the control/status bypass enforces the security policy that relates to the unencrypted system level information that crosses the Red/Black Boundary. A waveform may have a security policy as well. An example of a waveform security policy is when the housekeeping information crosses the Red/Black boundary over the protocol bypass.
- Load Security Policy

5.4.2 Access-Control Database (Security Policy).

Description: The access-control database is the element of the system that embodies the security policy. Different system security policies can be supported by changes to the access control database without modification of the access control mechanism.

The JTRS use a “flow control” security policy. A flow control security policy is used because of its compatibility with the object oriented data model. A more traditional security model, such as the Bell and LaPadula model, uses the concept of passive objects and active subjects. The flow control model uses objects and messages. An object in the flow control model may possess properties of a passive data container, and an active agent. Messages become the method of transferring information between objects. The flow control model fits well with the object oriented design process and the information flow within the JTRS.

Architecture/Implementation: Both. The use of access control databases in conjunction with access control mechanisms is an architectural feature. The implementation of the access control database is left to the developer.

Place in the Architecture: The operating environment supports and enforce the system security policy. Other security policies may exist at a more local level, such as the Control/Status Bypass.

Implied Design:

- The operating environment implements the access-control database for system control and message routing functions.
- The operating environment contains the necessary information that identifies the software and hardware resources necessary to establish non-core applications without violating the security policy.
- The operating environment contains the necessary information required to provide message registration (establish virtual communication channels) between those resources that provide the non-core applications.
- The JTRS uses a "flow control" security policy.

System High Access Control Database

- The system high access-control database supports single level objects on the Red-side of the JTRS.
- All possible (allowable) message traffic is identified within the security policy in the operating environment.

MILS Access Control Database

- The MILS access control database includes labels that include security classification and security compartmentalization.
- Each GPP permits applications at a single security classification level within a single compartment (if any compartments exist).
- Single classification level enforcement on each GPP is provided by the security policy within the operating environment.
- The security policy permits only single level objects within the system. Objects include software, hardware, and input/output devices (e.g., Ethernet port).

5.4.3 Intrusion Tools.

Description: Intrusion detection allows the system to identify when an unauthorized user has gained access to the system, or an authorized user has gained access to a portion of the system not within their privileges. Intrusion detection is a sub-class of access tools.

Architecture/Implementation: Implementation.

Place in the Architecture: Not applicable.

Implied Design:

- Rules establishment and Audit

5.4.4 Virus Detection Tools.

Description: Virus detection allows the system to identify when software executables have been altered or added maliciously.

Architecture/Implementation: Implementation.

Place in the Architecture: Not applicable.

Implied Design:

- Verify software integrity (e.g. CRL, secure Hash)

5.5 SYSTEM APPLICATION FUNCTIONS.

5.5.1 Declassified Box.

Description: A declassified box is a terminal that has been placed in the Cryptographically Controlled position.

Architecture/Implementation:

Place in the Architecture:

Implied Design:

- Key splits
- Generate Random
- Erase Red memory
- Zeroize Red memory
- Erase Classified application

5.5.2 System Startup.

Description: A JTRS performs self-test on all processors within the system. The processors report test status to the radio self-test application. Cooperative tests between processors can then be initiated. Each processor reports the cooperative tests results to the radio self-test application. The JTR radio becomes ready for application loading after it is determined sufficient radio assets exist.

Architecture/Implementation: Each architectural element performs its startup test sequence.

Place in the Architecture: Core framework or application

Implied Design:

- ▪ Normal (Token in, Switch ON)
- Initialize (BOOT) System Test
- Update Token

- Pr-Load Configuration file (punch in or call file waveform)
  - Match I/O ports to waveform
  - Load Parameters
- Idle channel
- Run
- Monitor
- Store tables and applications
- Idle
- Shut down (normal return to off)

Part of shut down process (Normal, During normal shut down box gets declassified)

- Store Tables and applications
- Erase RED memories and applications
- Overwrite RED algorithms
- Zeroize RED key
- Zeroize all TRANSEC keys
- Remove switch key

If Token is removed (Switch still in ON position)

- Overwrite RED Algorithm
- Zeroize RED keys
- Erase RED memories
- Disable Decrypt
- Inhibit CT outputs
- Zeroize all TRANSEC Keys

Standby

- No TX or RX
- Power Savings (only power necessary functions)
- Declassify
- Capability to retain Key(s)
- Hot Boot (could be classified as standby)

Power down (Anomalies)

- Under voltage detection

- Transient detect
- Tamper Detect

5.5.3 Identify Resources (Status Available Algorithms/Key Configuration Management).

Description: The JTRS radio configures virtual channels based on resource availability. Each architectural component provides the resources available to the *DeviceManager* upon startup. Unique INFOSEC capabilities includes encryption/decryption algorithms available, Tag generator and traffic, and key encryption keys available (TEKs and KEKs). The TEKS and KEKs includes a reference to the algorithm where they are used.

Architecture/Implementation: The JTRS radio determines the algorithm and key material available upon system startup. This information is provided to the Domain Manager by the CS/S subsystem. Changes to the algorithm and key material status is also reported to the Domain Manager. The CS/S supports queries by the Domain Manager of the current algorithm and key data available.

Place in the Architecture: Core Framework

Implied Design:

- The CS/S is capable of determining current resources available at startup.
- Failure to identify resources causes an alarm condition to be generated.
- The CF generates an alarm if the resource status information is not provided by the subsystems.
- The CS/S function is capable of identifying algorithm and key material when data is located in bulk storage.
- Initialize
- Configure

5.5.4 Pick Algorithm – Mode of Algorithm.

Description: Several of the algorithms used within the JTRS have multiple modes. The performance requirements for waveform operation require that the algorithm mode be switched without having to reload an algorithm to achieve a different mode of that algorithm.

Architecture/Implementation: Programmable encryption devices, used within the JTRS architecture is configured with all modes of a specific algorithm on each virtual channel that is instantiated. This implementation allows switching of algorithm modes.

Place in the Architecture: N/A

Implied Design:

- Modes of the algorithm can be switched on a message to message basis.

5.5.5 Frequency HOPSETs and LOCKOUT Sets.

Description: Applications on the black side of the JTRS architecture require the use of frequency information to tune the RF. Frequency HOPSETs specify collections of frequencies to be used by a waveform net. Lockouts specify portions of the band that are not to be used by the waveform net. A lockout set is directly correlated with a frequency hopset.

When a virtual circuit is created, the circuit have associated with it a hopset and optionally a lockout set. This information is traditionally loaded into a radio using the Fill Port Interface. The JTRS architecture continues to support this implementation, directing the frequency fill information from the CS/S subsystem to the Black Subsystem.

Architecture/Implementation: The CS/S subsystem accepts the input of frequency data to the fill port. The CS/S subsystem provides the frequency fill information to the Black Side Link Application. Distribution and control of the frequency information is performed on the Black side of the radio.

Place in the Architecture: Black Application, CS/S subsystem, File System

The JTRS architecture provides the interface for passing the hopset and lockout data outside of the INFOSEC boundary.

Implied Design:

- DS-101/102 fill interfaces is used to load fill information.
- The CS/S manages the identification and distribution of fill information to the red and black applications.
- The radio operator is required to place the radio into the fill mode prior to initiation of the fill process.
- The radio operator provides the label used to associate the frequency information with a net.
- The radio operator indicates whether the fill information is intended for immediate use or is intended for bulk storage destination.
- The radio maintains frequency parameters for a duration consistent with the storage requirements of cryptovariables (72 – 144 hours).
- The architecture supports the loading of HOPSETs and LOCKOUT Sets via a user interface.
- User Interface
- Handle classified Data

5.5.6 Establish Waveform Parameters.

Description: A set of parameters is required to complete a waveform instantiation. (These parameters includes a net identifier, keys, lockouts etc.) An operator may select these parameters at the operator GUI or select a configuration file that contains the required information.

Architecture/Implementation: After the resources have successfully been allocated to a waveform, parameters are passed to the application. These parameters are entered by the operator on a per channel basis.

The architecture supports the parameter selection required for waveform instantiation. The architecture supports the operator or file parameter configuration.

Place in the Architecture: Domain Manager, Application profiles

Implied Design:

- Waveform parameters are established after the Domain Profile includes successful association of the resources to the waveform. These resources encompass red, black, and CS/S architectural elements.
- Creation and establishment for configuration files.

5.5.7 Establish Data Path.

Description: The JTRS radio establishes circuits by connecting resources. These circuits establish the data path for a given channel. (E.g. DAMA with a Teletype user I/O)

Architecture/Implementation:

Place in the Architecture: Domain Manager

Implied Design:

- Domain Manager establishes the circuit.
- Build/Establish the file for execution by the Core Framework

5.5.8 Security Monitor.

Description: The Security Monitor function provides added assurance to the proper operation of hardware relative to the execution of software.

The following functions are examples of monitor logic:

Software Flow Monitor – monitor execution of proper instruction sequences for instructions involved in critical operations such as data movement to and from the encryption device and during subsystem health checks.

Data Flow Monitor – monitors specific address ranges being accessed by the processor and by locking out memory operations under the detected failure conditions.

Software Flow and Data Flow Alarm Check Logic – generates simulated error cases for Software and Data Flow monitors to detect as part of the Alarm/Alarm Checking process.

Alarm Restriction/Notification Logic - restricts operations and notifies alarm conditions to the Red and Black interfaces.

Architecture/Implementation: The Security Monitor Function is within the INFOSEC boundary and RED subsystem. Architecturally, the extension of this concept, to provide Software Flow Monitoring and Data Flow Monitoring in support of MLS, would require the Monitor Function affected red subsystems as a minimum.

Place in the Architecture: INFOSEC boundary, Red Subsystems.

Implied Design:

- Monitoring of software functions is required to provide redundancy.
- The security monitor supports the enforcement of separation and execution.

5.5.9 Function: Status.

Description: All subsystems generate status information. The JTRS radio provide a function for collection of status in order to determine health of resources. Status reports indicating resource failure are used to restrict operation of the radio. Status is reported at power up and throughout the power-on state of the radio.

Architecture/Implementation: The JTR architecture supports the collection of status information.

Place in the Architecture: Domain Manager/Application

Implied Design:

- Several elements of the Core Framework works in conjunction with each other to remove failed channels.
- Faults detected results in restriction of radio operations.
- Status is monitored at all times. Corrective action is initiated when a failure is reported.
- Each subsystem monitors its own status.

  The monitor function reports status to a centralized status monitor.
- The monitor function reports all Alarms (e.g. PTD power, TAMPER, alarm…).

The status function classifies failures and associate actions with failures in accordance with loaded security policy. The subsystem status function initiates restriction of local operations when the failure classification indicates this is the proper response. The centralized status function initiates status response when a failure is reported. Failures reported is logged.

5.5.10 Built-In Test.

Description: Built-In Test (BIT) is performed at power-on in order to allow the subsystems to develop a healthy resource list. In addition to containing subsystem tests, there is a need for cross-subsystem tests, to insure that the elements required to instantiate a thread are available. Cross-subsystem tests are controlled from a central location.

Background tests are performed continuously, with status generated when an error is detected. Built-In Test can be initiated by the operator. The operator has the capability to select the level of testing to be performed.

Architecture/Implementation: Implementation

Place in the Architecture: All subsystems contained within the JTRS radio. Domain Manager (application).

Implied Design:

- Failure of resources results in restriction of radio operations.
- Operator commanded Built–In Test requires the operator to place the radio into a test mode (no traffic).
- Background BIT tests is non-intrusive to normal operation of the radio.
- Instantiated channel test

BIT is performed when power is applied to a subsystem. BIT status is reported to a central BIT control function for the radio. The central BIT function initiates cross-subsystem tests once the individual subsystem tests have been performed. BIT results are used to populate the resource tables at power-on. Subsequent detection of faults shall update the resource tables. BIT results are logged by the radio.

5.5.11 Human Computer Interface.

Description: The HCI function provides access control to the radio. Because different users have different roles with respect to the radio, the access control mechanism includes a segmentation of operations allowed.

The HCI provides mode control to the radio. The HCI provides the operator the ability to select and instantiate a waveform.

Visual alarms are sent to the HCI for display on the user console. Audio alarms are also sent to the HCI via an audio interface.

Architecture/Implementation The architecture must support the messaging between the HCI and the system. HCI is located on the RED or BLACK side of the radio. The HCI supports the operator for radio configuration and human data entry, which includes biometric elements.

Place in the Architecture: The HCI mechanism is available to a local user or a remote user. Domain Manager – Core Services Access Control *(new)*, CORBA interface – Message.

Implied Design:

- Use of access control mechanism based on Token implies a database of valid users must be maintained and distributed.
- Maintenance and distribution of a Token privilege.
- The architecture is the authentication of a user, that list can be stored in a protected fashion, and that the list can be updated to add and remove users by a system administrator.
- Subsystems report alarm conditions that result in display or audible alarm to the HCI.
- The radio locks out a user who is not authenticated.
- The lockout method must be able to be cleared by a user with appropriate privileges.
- Authentication of remote user/commands.

5.5.12 Encrypt/Decrypt storage.

Description: The CS/S needs to provide bulk/selective en/decryption services for user apps or CF.

Architecture/Implementation: Impacts CS/S Messaging with the CF. CF needs to allow this type of messaging.

Place in the Architecture: The en/decrypt services of the CS/S is capable of bulk en/decryption as required.

Implied Design:

- Look at implied requirements on Encrypt/Decrypt.

5.5.13 Storage Commands – Human Control Interface.

Description: The SCAS needs to support messaging so that the operator can perform the following functions:

Load fill, Load algorithms, select fill, select algorithm, view CS/S resource list of fill and algorithms, etc.

Architecture/Implementation: The CF supports the messaging required for the CS/S /HCI functionality.

Place in the Architecture: Radio Control

Implied Design: None

5.5.14 Storage Control – Verify (CS/S Configuration Control).

Description: The JTR must have a methodology for assuring security critical control has been accomplished with the proper notification to the user. A continuos check of this control is also required.

Architecture/Implementation: Need for a control validation object exists while the JTR is running. Storage control validation is a security function that constantly checks that the current control matches the requested control.

Place in the Architecture: Configuration Control resides with the CS/S or with the Domain Manager.

Implied Design: None

5.5.15 Storage Control – Process (CS/S Configuration).

Description: The JTR must have a methodology for assuring security critical control has been accomplished. Security critical control includes selection of algorithm and keys. It also includes selection of a plain text bypass function and the command bypass features that need to be validated.

Architecture/Implementation: A control validation object needs exists when the JTR is running. It is a security function that constantly checks that the current control request is allowable by the operator.

Place in the Architecture: Process configuration resides with the CS/S or with the Domain Manager.

Implied Design: None

5.6 SYSTEM FUNCTIONS.

5.6.1 Establish/Maintain Path (Security Related).

Description: A JTR is multi-channel capable. Each channel operates independently. Each channel is established as a virtual connection between User I/O, Red Applications, CS/S, Black Applications, Modem and RF.

Architecture/Implementation: A JTR is capable of instantiating a channel that establishes a virtual path between the User I/O, Red Applications, CS/S, Black Applications, Modem and RF.

Place in the Architecture: Core Framework/Application?

Implied Design:

- The establishment of virtual channels is independent of the Hardware Architecture.

5.6.2 Memory.

The SCAS is defined to keep as much of the security functionality out of the Core Framework as possible. It uses COTS S/W were applicable to take advantage of technological improvements and promote commercial acceptance of the architecture. Therefore, it is desirable to have all memory functions that are security critical, be executable applications.

5.6.2.1 Memory Allocation (Security Related).

Description: Before a waveform is instantiated, the radio checks for sufficient memory at all processing nodes. This behavior is done in the Domain Management function.

Architecture/Implementation: Prior to waveform instantiations, the *DeviceManager* needs the target hardware memory profile. The Memory Management Unit, (MMU), maintains the memory map, which delineates used and unused memory.

Place in the Architecture: Operating System (OE)/Application

Implied Design:

- The destination H/W needs to identify various assets that are available.
- The amount of usable memory is one of these assets.

5.6.2.2 Memory Separation.

Description: There are two kinds of memory separation. Memory can separate executable objects or traffic data.

Architecture/Implementation: Architecture must guarantee ownership of memory space for executable objects and/or tasks that exits in a memory space. A rule set defines constraints. The architecture must notify the operator of any rule violation.

Place in the Architecture: All red side processing nodes must be capable of separating executable objects and traffic data.

Implied Design:

- None.

5.6.3 Memory Erasure (Object Reuse).

Description: The reallocation or tear down of an instantiated algorithm. It also refers to the elimination of red traffic data from red applications.

Architecture/Implementation: Implementation.

Place in the Architecture: CS/S and Red traffic data processing on all red processing nodes.

Implied Design:

System High Operation

- The Red side zeroizes its memory if the CS/S has signaled it to do so.

MILS Operation

- A Red GPP zeroizes memory used to process data at one classification level prior to reusing it to process data at a lower classification level.

5.6.4 Power.

Primary power is filtered to prevent EMC/EMI radiation.

5.6.4.1 Power Red/Black.

Description: The Radio is a well-defined Red/Black boundary.

Architecture/Implementation: The Power Subsystem, (S/S), is able to isolate logical Red noise from the Black power.

Place in the Architecture: The requirement primarily resides in the Power S/S. (May change if local power regulation is used.)

Implied Design: None

5.6.4.2 (Power) Voltage.

Description: Independent voltage detection scheme to protect the cryptographic logic from operating during an over voltage, (OV), or under voltage, (UV) event.

Architecture/Implementation: OV/UV protection is provided on all voltages for security critical logic. Upon detection of an OV/UV event, the SCAS supports a quick shutdown of all cryptographic critical functions.

Place in the Architecture: CS/S

Implied Design:

- An OV/UV event must have priority over pending tasks. Nothing in the architecture can stop the shutdown.
- Notification to the HCI.
- Log the event.

5.6.4.3 Protection (Power).

Description: The power subsystem needs to protect the radio from power surges, spikes and dropouts.

Architecture/Implementation: The Power subsystem protects the radio from damage in the event of power surges, spikes and dropouts.

Place in the Architecture: Power subsystem

Implied Design:

- Notification to the HCI.

5.6.5 Parameter Storage.

Description: Storage of any system black data. Any subsystem, including CS/S needs the ability to store black information in bulk storage.

Architecture/Implementation: Bulk storage for black information is provided through the use of the file system.

Place in the Architecture: Core Framework

Implied Design:

- The CF supports a file system for storing and retrieving information.

5.6.6 TEMPEST.

Description: Undesirable Radio emanations

Architecture/Implementation: Implementation. Requirements established by a UIC.

Place in the Architecture: Distributed

Implied Design:

5.6.7 TAMPER.

Description: Unauthorized JTR entry.

Architecture/Implementation: AJTR includes the detection and reaction to unauthorized intrusions.

Place in the Architecture: Distributed

Implied Design:

5.6.8 Physical Security.

<u>Description:</u>

<u>Architecture/Implementation:</u>

<u>Place in the Architecture: Mechanical</u>

<u>Implied Design:</u>

6 COMMON SERVICES AND DEPLOYMENT CONSIDERATIONS.

6.1 COMMON SYSTEM SERVICES.

There has been no common services identified at this time. Therefore, no rationale has been generated.

6.2 OPERATIONAL AND DEPLOYMENT CONSIDERATIONS.

There has been no common interfaces or features identified at this time. Therefore, no rationale has been generated.

7 ARCHITECTURE COMPLIANCE.

This section provides linkage between operational requirements of the JTRS ORD 23 March 98 and architecture requirements in the SCAS. These are very different types of requirements documents: the ORD is motivated by the needs of operational forces and identifies capabilities required for future communications. The SCAS constrains the design of systems that are intended to meet these operational requirements by reducing the cost of applications software portability and readily permit upgrades to new waveforms to meet future operational needs.

The tables list ORD requirements (Basic ORD and Domain Annexes) and JTRS Program Objectives, show their category (KPP, Threshold, Objective, or Not Labeled), list those that were identified as possible architecture drivers in Step 1 of JTRS (requirements that engineering experience indicated might be difficult to implement in a JTRS Architecture constrained design), identify "architecture influencers" assess the influence (high, medium or low); and point to the SCAS section or paragraph that pertains to the requirement. (see definitions below).

The final column provides rationale for the linkage from two viewpoints:

how the requirement caused or directly relates to the SCAS element, and

given the SCAS requirement statement, how does it meet (and/or allow to be met) the ORD requirement.

This column is filled for all requirements identified as influencing the architecture.

The following definitions were used:

Architectural Influencers – Requirements from the JTRS ORD or JTRS Program objectives from the Step 1 BAA that caused features of the architecture to be included. For example the program objective of "S/W independence from H/W" influenced the architecture to include the CF, CORBA and POSIX to isolate the software from the hardware processor.

Performance Difficulty – an engineering assessment, based primarily on experience with legacy implementations, of the difficulty that an implementer might have in meeting an ORD requirement using the JTRS architecture. Assessments are either "Low", "Medium", or "High".

Table 7-1: ORD Architecture Drivers

ORD						
ORD Paragraph Number	OPERATIONAL REQUIREMENTS DOCUMENT (ORD) For JOINT TACTICAL RADIO (JTR)	Requirement Category - KPP or Threshold	Identified as High Arch Driver in Step 1	Architecture Driver: High, Med, Low	SCAS Reference	SRD TEXT
4.a.(1)(a)	The JTR architecture is capable of supporting secure and non-secure voice, video and data communications using multiple narrow-band and wideband waveforms as specified in paragraph 7, tables 1 and 2, identifying threshold KPP, threshold, and objective requirements for FY'00 through FY'05, including future waveforms as they are developed.	KPP	Yes	High	Sec 2.2.2.2 (footnote); 2.2.1.7.5; 3.1.3.2.2.5.1.3; Sec 5 (V 2.0) and Security Supplement	This requirement added security examples in section 2 and 3. Sec 5 of the SCAS contains requirements that are identified as required for the operating environment. The Security Supplement contains APIs for the Security entity. These elements standardardize the JTRS approach for security implementation and be used in the secure modes of the ORD waveforms.
4.a.(1)(b)	The JTR program provides an internal growth capability through an open systems architecture approach.	KPP	Yes	Low	Sec 4.5.1, 4.5.2, 4.5.2.2, JTRS API Supplement	The SCAS is based on multiple open system standards These make up the OE. In those instances where a vender selects from a particular standard hardware interface and includes unique connections between hardware modules, the SCAS requires those to be published so that others may reuse the hardware. API definitions establish standard interfaces available for application software. Each compliant software application publishes the interfaces to the operating environment and makes them available for reuse for other applications and upgrades.
4.a.(1)(b)	The growth capability is in compliance with the Joint Technical Architecture (threshold) (KPP).	KPP	Yes	Low	Sec 1.2.1	The OE provides an architectural framework for a JTA system.
4.a.(1)(b)	The growth capability is modular, scaleable, and flexible in form factor (threshold) (KPP).	KPP	Yes	Med	Sec 3.1.1 3.2, 4.5	4.5 General hardware rules do not restrict form factor, either at the module level or at the chassis level, to permit flexible implementation to fit the specific needs of procuring agency. 3.1 & 3.2 The software architecture (Operating system, CORBA, Core Framework, and application requirements) collectively provides a modular and scalable environment for software applications. The use of COTS, both hardware and software, aids in the domain's ability to be both modular and scalable.

ORD						
ORD Paragraph Number	**OPERATIONAL REQUIREMENTS DOCUMENT (ORD) For JOINT TACTICAL RADIO (JTR)**	Requirement Category - KPP or Threshold	Identified as High Arch Driver in Step 1	Architecture Driver: High, Med, Low	SCAS Reference	SRD TEXT
4.a.(1)(c)	The JTR provides the operator with the ability to load and/or reconfigure modes/capabilities (via software) while in the operational environment (threshold)(KPP).	KPP	Yes	High	3.1.3.4, 5.1.3, B.3.1.1, B.3.1.2, D	Radio reconfiguration in the operational environment utilizes application interfaces such as the installer utility with its attendant CF *DomainManager*, *DeviceManager*, and *File* Manager with specific POSIX behavioral single-process and multi-processes. The concept and implementation of the Domain Profile are direct results of this requirement.
4.a.(1)(d)	The JTR has the ability to be reconfigured (hardware changes/upgrades) in the operational environment (threshold).	Threshold	Yes	Low	4.5.1, 4.5.2, 4.5.4, 3.2.2.1.1, D3	4.5.2 Published definition of hardware critical interfaces allows hardware modules to be replaced in an operational environment with other hardware modules meeting those published interfaces. 4.5.1 *Device* profiles provide hardware characteristics for registration with the core framework. 4.5.4 Hardware modularity facilitates replacement in an operational environment. Hardware configurations of a domain are maintained by their Hardware profile, Device Package Descriptor D3. These profiles are administered by the installation utility of the SCAS.
4.a.(1)(e)	The JTR is capable of operating in a radio frequency spectrum from 2 MHz to 2 GHz suitable for the particular set of waveforms that the JTR support (threshold) (KPP).	KPP	Yes	No		
4.a.(1)(e)	The JTR is capable of incorporating military and commercial satellite and terrestrial communications above 2 GHz (objective).	Objective		No		
4.a.(1)(f)	The JTR has the ability to retransmit/cross-band information between frequency bands/waveforms supported (threshold)(KPP) Maritime/Fixed Station Domain (Objective).	KPP	Yes	Med		The SCAS does not define performance for specific waveforms or capabilities.

ORD						
ORD Paragraph Number	OPERATIONAL REQUIREMENTS DOCUMENT (ORD) For JOINT TACTICAL RADIO (JTR)	Requirement Category - KPP or Threshold	Identified as High Arch Driver in Step 1	Architecture Driver: High, Med, Low	SCAS Reference	SRD TEXT
4.a.(1)(g)	The JTR is capable of operating on multiple full and/or half-duplex channels at the same time (threshold)(KPP).	KPP	Yes	High	3.2, B.3.1.2	RF channels, whether half or full duplex, are instantiated by waveform applications as defined in SCAS section 3.2. The number and availability of channels is governed by the ORD-specified domain. Simultaneous operation of multiple channels becomes a function of the performance of the domain hardware. A multi-process OS is defined by the SCAS and is detailed in appendix B.3
4.a.(1)(h)	The JTR has the capability of automatic protocol conversion and message format translation of voice, video, or data between frequency bands or waveforms as specified in paragraph 7, Table 1 (threshold).	Threshold	Yes	No		
4.a.(1)(i)	Without interfering or overriding terminal operations, the JTR is capable of distributing and accepting software upgrades that have integrity and can be authenticated when transmitted through the network with which it interfaces (threshold).	Threshold	N	High	Sec 5.1.3; 4.2.3.5	The SCAS provides for both the Domain Manager and the physical interface that would use it. The interface is specific to the target domain and platform. The SCAS does not define performance required to meet the non-interference requirement. That depends on implementation specific hardware and software.
4.a.(1)(j) 1	comply with applicable National, and International spectrum management policies and regulations(threshold).	Threshold	N	No		
4.a.(1)(j) 2	be mutually compatible with other electric or electronic equipment within their expected operational environment (threshold).	Threshold	Yes	No		
4.a.(1)(k)	The JTR provides the ability to scan a minimum of 10 operator designated fixed frequencies or presets (threshold).	Threshold	N	No		

ORD						
ORD Paragraph Number	**OPERATIONAL REQUIREMENTS DOCUMENT (ORD) For JOINT TACTICAL RADIO (JTR)**	**Requirement Category - KPP or Threshold**	**Identified as High Arch Driver in Step 1**	**Architecture Driver: High, Med, Low**	**SCAS Reference**	**SRD TEXT**
4.a.(1)(l)	The JTR or its installation kit provides domain specific interfaces to ancillary equipment in order to minimize platform integration impact, i.e., power amplifiers, power supplies, antenna couplers, antennas, etc. as stated in the annexes (threshold).	Threshold	N	No		
4.a.(1)(m)	The JTR employs protective measures against electromagnetic pulse (EMP) and directed energy threats (objective).	Objective	N	No		
4.a.(1)(n)	After an unexpected power loss, and upon restoration of power, the JTR is capable of completing a components diagnostics test and a systems recovery to include: hardware, software, presets and settings (threshold).	Threshold	N	Low	Sec 3.1.3.1.3	The SCAS provides for testable objects that can perform any required diagnostics tests. The SCAS also provides for but does not describe an HCI Client that interfaces to the Domain Manager. The HCI can be implemented to retain knowledge of the operating state that enables it to direct the CF to configure to any known state. Knowledge of the nature of a power loss - expected vs. unexpected - is a HCI Client implementation decision and is not covered in the SCAS.
4.a.(1)(o)	The JTR provides a standard interface to exchange voice/video/data with Service host systems (threshold).	Threshold	N	Low	Sec 1.2.1; JTRS API Supplement	The JTA imposes standard interfaces for these data exchange types. No specific requirements on the external interfaces of the waveform application are included in the SCAS. However, the interface standards imposed by the JTA apply to JTRS as a C4I system.
4.a.(2)(a)	The JTR provides a scaleable, embedded programmable cryptographic capability, which support black key (threshold).	Threshold	N	Med	Sec 2.2.1.7.5; 4.2.3.4; JTRS Security Supplement	The SCAS does not define how INFOSEC requirements are to be implemented. It does isolate cryptographic functions so that programmable implementations can be used inside this entity. The JTRS Security Supplement incorporates the requirements for implementing black key fill.
4.a.(2)(b)	The JTR provides transmission security (TRANSEC) capability (threshold).	Threshold	N	No		

ORD						
ORD Paragraph Number	**OPERATIONAL REQUIREMENTS DOCUMENT (ORD) For JOINT TACTICAL RADIO (JTR)**	**Requirement Category - KPP or Threshold**	**Identified as High Arch Driver in Step 1**	**Architecture Driver: High, Med, Low**	**SCAS Reference**	**SRD TEXT**
4.a.(2)(c)	The JTR is capable of interfacing with an National Security Agency (NSA) approved electronic key management system (EKMS) (threshold).	Threshold	Yes	Low	Sec 4.2.3.4	The SCAS does not define performance for specific waveforms or capabilities. Specifically, EKMS extends well beyond the scope for any specific JTR product. The SCAS does define a hardware INFOSEC Class that has a key fill type attribute that permits implementation of hardware to interface in an EKMS compliant manner when such compliance is defined.
4.a.(2)(d)	The JTR is capable of using over the air rekeying (OTAR)/zeroizing (OTAZ)/ transfer (OTAT) as implemented by the Key Management Authority (threshold).	Threshold	Yes	Med	Security Supplement	This capability is defined in the JTRS security supplement. Implementation of this capability has NSA participation and conforms to NSA guidelines.
4.a.(2)(e)	The JTR is capable of remote identification and exclusion (lockout) (threshold).	Threshold	N	Low	3.2.1, 5.1.3	Remote identification and exclusion are a function of a waveform's ECCM capability. A JTRS channel is instantiated as that type of waveform by installing its application and executing it.
4.a.(2)(f)	The JTR is capable of supporting encrypted Global Positioning System (GPS) (threshold).	Threshold	N	Low	Sec 2.2.1.7.6 Figure 4-2	The SCAS provides for a GPS Hardware Child Class. The UtilityResource can provide an implementation-defined SitAwareResource Type that provides GPS-derived information to the waveforms. Future versions of the SCAS may define the API for the UtilityResource and the SitAwareResource
4.a.(2)(g)	After a primary power loss, the JTR is capable of retaining perishable cryptographic variables for at least 72 hours (threshold)	Threshold	N	No		
4.a.(2)(g)	After a primary power loss, the JTR is capable of retaining perishable cryptographic variables for at least 144 hours (objective).	Objective	N	No		
4.a.(2)(h)	The JTR is capable of implementing public cryptography to provide privacy (objective).	Objective	No	Low	Sec 2.2.1.7.5; 4.2.3.4; JTRS Security Supplement	Cryptographic functions including algorithms used and key fill method are isolated in the SCAS to the INFOSEC entity within the architecture. Specific implementations of the type of key and cryptographic method used are not part of the SCAS, but are permitted in these implementations. Security policy and requirements are specified in the JTRS Security Supplement to v 2.0

ORD						
ORD Paragraph Number	OPERATIONAL REQUIREMENTS DOCUMENT (ORD) For JOINT TACTICAL RADIO (JTR)	Requirement Category - KPP or Threshold	Identified as High Arch Driver in Step 1	Architecture Driver: High, Med, Low	SCAS Reference	SRD TEXT
4.a.(2)(i)	The JTR provides for INFOSEC and protection of data at multiple levels of Security from Unclassified through Secret (objective).	Objective	No	High	Security Supplement	MLS capability is defined in the JTRS security supplement. Implementation of this capability has NSA participation and conforms to NSA guidelines.
4.a.(2)(i)	The JTR provides for INFOSEC and protection of data in a Secret High network (threshold).	Threshold	Yes	Med	Security Supplement	This capability is defined in the JTRS security supplement. Implementation of this capability has NSA participation and conforms to NSA guidelines.
4.a.(2)(j)	In conjunction with waveforms for JTR domain family members in Paragraph 7, Tables 1 and 2, the JTR is capable of supporting cryptographic functions which operate with or have an external interface to cryptographic systems as listed in Annex D (threshold).	Threshold	Yes	Med	Sec 4.2.3.4	The SCAS does not define performance for specific waveforms or capabilities. The SCAS does define a hardware INFOSEC Class that has attributes sufficient for all required cryptographic functions.
4.a.(2)(k)	The JTR is capable of being handled as an unclassified Controlled Cryptographic Item (CCI) when the embedded device that provides security is not keyed (threshold).	Threshold	N	No		
4.a.(2)(l)	The JTR has the capability to zero locally by requiring at least two discrete operator actions to reduce the probability of accidental zeroing (threshold).	Threshold	N	No		
4.a.(2)(m)	The JTR employs the Defense Information Infrastructure (DII) Key Management Infrastructure in supporting JTR integrity, identification, and authentication requirements (threshold).	Threshold	N	Low	JTRS Security Supplement	The JTRS Security Supplement (to be updated in V1.1) imposes requirements derived from the DII Key Management Infrastructure. Material contained in the SCAS on key management has been coordinated with the NSA.

ORD						
ORD Paragraph Number	OPERATIONAL REQUIREMENTS DOCUMENT (ORD) For JOINT TACTICAL RADIO (JTR)	Requirement Category - KPP or Threshold	Identified as High Arch Driver in Step 1	Architecture Driver: High, Med, Low	SCAS Reference	SRD TEXT
4.a.(2)(n)	Each JTR is equipped with the capability to receive Global Positioning Satellite (GPS) signals. This GPS receive capability is in addition to the number of channels for each JTR category specified by the Domain Annexes.		N	Low	Sec 4.2.2, 4.5	The SCAS supports installation of GPS receive capability.
4.a.(3)	The JTR is capable of providing scaleable networking services for connected RF (over the air) networks, host networks, and hybrid networks in Increment 3 and in accordance with the phased procurement implementation specified in paragraph 7 (threshold) (KPP).	KPP	N	Med	Sec 2.2.2; JTRS API Supplement	Networking requirements are met by the operating environment and application software running in that environment and are not specified within the SCAS. The SCAS does standardize APIs for network applications and can host waveforms that have scalable networking capability (e.g., VRC-99)
4.a.(3)(a)	The networked JTR extends, between and across the geographical and/or organizational boundaries within a nominal area of operations (threshold) (KPP).	KPP	N	No		
4.a.(3)(b)	The networked JTR provides a scaleable and interoperable means to establish point-to-point (two way), multi-point (two way), multicast (up to 100 selected nodes), and broadcast data capability between/among any user-selected nodes in a joint network (threshold).	Threshold	N	Med	Sec 2.2.2; API Supplement	Networking requirements are met by the operating environment and application software running in that environment and are not specified within the SCAS. The SCAS does standardize APIs for network applications and can host waveforms that have scalable networking capability (e.g., VRC-99)
4.a.(3)(c)	The networked JTR provides for mobile JTRs to readily transfer between authorized networks. This transfer should be transparent to the user. (threshold).	Threshold	N	Low	3.2.1, 3.2.2.1	3.2.1 The SCAS supports the hosting of various networking protocols as applications. 3.2.2.1 Seamless transfer of data is supported through the use of the Service API's. Service API's include Network, Security and I/O services for the domain.
4.a.(3)(d)	The networked JTR provides routing capability, interface connectivity that extends into the Internet Protocol, military packet networks (threshold).	Threshold	N	No		
4.a.(3)(d)	The networked JTR provides routing capability, interface connectivity that extends into the Internet Protocol and/or cell networks (objective).	Objective	N	No		

ORD						
ORD Paragraph Number	OPERATIONAL REQUIREMENTS DOCUMENT (ORD) For JOINT TACTICAL RADIO (JTR)	Requirement Category - KPP or Threshold	Identified as High Arch Driver in Step 1	Architecture Driver: High, Med, Low	SCAS Reference	SRD TEXT
4.a.(3)(e)	The networked JTR performs dynamic intra-network and inter-network routing for data transport based on priority (threshold).	Threshold	N	No		
4.a.(3)(f)	The networked JTR includes hardware and software sufficient to organize, manage, and dynamically control network connectivity structures, routing mechanisms, and bandwidth allocations (threshold).	Threshold	N	No		
4.a.(3)(g)	The networked JTR selectively transmits individual location information to selected JTR nodes.	Threshold	N	No		
4.a.(3)(g)	The networked JTR transmission is passed in the Military Grid Reference System and/or in the latitude and longitude to a host system (threshold).	Threshold	N	No		
4.a.(3)(h)	The JTR system provides network management capability to respond to changes in mission or organization, and reconfigure at a minimum a battalion-sized joint task force network (with approximately 150 JTRS terminals), in 15 minutes (threshold, KPP).	KPP	N	Low	Sec 1.2.1; JTRS API Supplement	JTRS implementations are C4I systems that complies with JTA data exchange and network management protocols. These protocols provide for network link status. Applications waveforms that provide wireless networking capability is required to support network management requirements. These requirements are not part of the SCAS, but would be application waveform performance requirements.
4.a.(3)(h)	The JTR system provides network management capability to respond to changes in mission or organization, and reconfigure at a minimum a battalion-sized joint task force network (with approximately 150 JTRS terminals), in 5 minutes (objective).	Objective	N	Low	3.1.2, 3.1.3.4, D	CORBA software provides rapid system reconfiguration. The Domain Profile implementation provides reconfigurability capability.
4.a.(3)(i)	The JTR network has the capacity to meet the information flow of waveforms/capabilities as specified in paragraph 7, Table 1 (threshold).	Threshold	Yes	High	Sec 3.2.2.1.2 and Figure 3-35.	The SCAS does not define performance required to meet waveform/capabilities. That depends on implementation hardware. Stressing waveforms are identified in this SRD section and rationale provided that shows these can be implemented using the SCA with today's technology. The SCAS permits alternate transfer paths if middleware/overhead limits performance.

ORD						
ORD Paragraph Number	**OPERATIONAL REQUIREMENTS DOCUMENT (ORD) For JOINT TACTICAL RADIO (JTR)**	**Requirement Category - KPP or Threshold**	**Identified as High Arch Driver in Step 1**	**Architecture Driver: High, Med, Low**	**SCAS Reference**	**SRD TEXT**
4.a.(3)(j)	The JTR provides information to Service and joint network management tools to assess and report network link status (threshold).	Threshold	N	Low	Sec 1.2.1; JTRS API Supplement	JTRS implementations are C4I systems that complies with JTA data exchange and network management protocols. These protocols provide for network link status. Applications waveforms that provide wireless networking capability is required to support network management requirements. These requirements are not part of the SCAS, but would be application waveform performance requirements.
4.a.(3)(k)	The JTR network provides a name-to-address translation service that supports automatic registration and de-registration of host names and addresses (threshold).	Threshold	N	No		
4.a.(3)(l)	The JTR network provides the capability for users to address data to other users by using position/ organization names in the address fields (e.g., S3.2AR.BDE) (threshold).	Threshold	N	Low	Sec 1.2.1; JTRS API Supplement	JTRS implementations are C4I systems that complies with JTA data exchange and network management protocols. These protocols provide for network link status. Applications waveforms that provide wireless networking capability is required to support network management requirements. These requirements are not part of the SCAS, but would be application waveform performance requirements.
4.a.(3)(m)	The JTR system provides the means to support message delivery based on geographic areas (objective).	Objective	N	Low	3.2.1, 3.2.2.1	3.2.1 The SCAS supports the hosting of various networking protocols as applications. Transfer of data over geographical areas is a function of the waveform application. 3.2.2.1 Application's are supported with Service API's which provide I/O and networking services.
4.a.(3)(n)	The JTR network provides information and be interoperable with the joint network management tool, to allow network managers to remotely identify and configure user access and profile parameters to prioritize users' network access and message delivery (threshold).	Threshold	N	Low	Sec 1.2.1; JTRS API Supplement	JTRS implementations are C4I systems that complies with JTA data exchange and network management protocols. These protocols provide for network link status. Applications waveforms that provide wireless networking capability is required to support network management requirements. These requirements are not part of the SCAS, but would be application waveform performance requirements.
4.b	The JTR has the following mission-capable requirements for both wartime and peacetime.					

ORD						
ORD Paragraph Number	**OPERATIONAL REQUIREMENTS DOCUMENT (ORD) For JOINT TACTICAL RADIO (JTR)**	**Requirement Category - KPP or Threshold**	**Identified as High Arch Driver in Step 1**	**Architecture Driver: High, Med, Low**	**SCAS Reference**	**SRD TEXT**
4.b(1)	The JTR has an operational availability (Ao) of 96 percent (threshold)(KPP).	KPP	N	No		
4.b(2)	The JTR hardware size and weight is compatible with current platforms, as specified in the domain annexes (threshold).	Threshold	N	No		
4.b(2)	The JTR hardware size and weight also consolidate many individual functions of current terminals into one physical chassis, thereby reducing weight and space requirements (threshold).	Threshold	N	No		
4.b(3)	The JTR is logistically supportable within each Service (threshold).	Threshold	N	No		
4.b(4)	The JTR internal test and diagnostic built-in-test (BIT) provisions is capable of fault isolation (threshold).	Threshold	N	No		
4.b.(1)	The JTR has an operational availability (Ao) of 99 percent (objective).	Objective	N	No		
4.c.(1)	The JTR provides an operator-selectable capability to operate in listening silence (receive only) mode (threshold).	Threshold	N	No		
4.c.(2)	The JTR effectively operates in wartime operations and in worldwide conditions as specified in the domain annexes (threshold).	Threshold	N	No		
4.c.(3)	The JTR is capable of being operated and maintained in a nuclear, biological and chemical (NBC) environment, as specified in the domain annexes (threshold).	Threshold	N	No		

ORD						
ORD Paragraph Number	**OPERATIONAL REQUIREMENTS DOCUMENT (ORD) For JOINT TACTICAL RADIO (JTR)**	**Requirement Category - KPP or Threshold**	**Identified as High Arch Driver in Step 1**	**Architecture Driver: High, Med, Low**	**SCAS Reference**	**SRD TEXT**
4.c.(4)	The JTR is capable of providing a platform- specific human computer interface, as specified in the domain annexes (threshold).	Threshold	N	Low	Sec 2.2.1.7.6; 3.1.3	The SCAS isolates user and platform interfaces through modularity of the software and hardware associated with those interfaces. Platform unique interfaces are accommodated within the User I/O hardware class and through extensions of the CF Resource class and the *Device* class such as I/O Device
4.c.(5)	The JTR is capable of incorporating power management to achieve maximum efficiency (threshold).	Threshold	N	No		
4.c.(6)	The JTR is capable of withstanding power surges (threshold).	Threshold	N	No		
5	Program support for the JTR is in place when the initial operational capability (IOC) is achieved.		N	No		
5	Program support for the JTR is expanded, as necessary, for each Service prior to achieving full operational capability (FOC).		N	No		
5.a.	Operator level maintenance is limited to reconfiguration for needed capabilities.		N	No		
5.a.	Preventive/corrective maintenance is limited to the predetermined lowest repairable unit (LRU).		N	No		
5.a.	Life-cycle logistics support factors is implemented that provide for cost effective maintenance of the JTR.		N	No		
5.b.	Where the DoD logistics structure is used, General Purpose Electronic Test Equipment (GPETE) is selected from existing standard GPETE equipment lists.		N	No		
5.b.	The use of Special Purpose Electronic Test Equipment (SPETE), special purpose support equipment, and special tools is avoided to the maximum extent possible.		N	No		

ORD						
ORD Paragraph Number	OPERATIONAL REQUIREMENTS DOCUMENT (ORD) For JOINT TACTICAL RADIO (JTR)	Requirement Category - KPP or Threshold	Identified as High Arch Driver in Step 1	Architecture Driver: High, Med, Low	SCAS Reference	SRD TEXT
5.b.	JTR BIT/built-in-test equipment (BITE) performance is not accomplished using GPETE, SPETE, or substituting modules.		N	No		
5.b.	For all non-developmental item (NDI) equipment the contractor identifies automatic test equipment (ATE).		N	No		
5.b.	For contractor provided logistics support, equipment and processes is identified.		N	No		
5.c.(1)	JTR must be easily maintainable and operable, incorporating the principles of modularity and commonality.		Yes	Low	Sec 4.5, 3.2	The SCAS hardware rules set supports this requirement from the hardware standpoint. The required use of CORBA software supports this from the software standpoint.
5.c.(1)	The JTR conforms to applicable human engineering design criteria.		Yes	No		
5.d.(1)	The JTR management and components provides checks for computer operations system viruses during systems initialization and routine operations.		N	Low		This capability is defined in the JTRS security supplement. Implementation of this capability has NSA participation and conforms to NSA guidelines.
5.d.(1)	The operator is alerted to a detected virus.		N	Low		This capability is defined in the JTRS security supplement. Implementation of this capability has NSA participation and conforms to NSA guidelines.
5.d.(2)	A software support capability is functional by JTR's Initial Operational Capability (IOC) and must provide for update, configuration control, and management of all computer programs and data.		N	No		
5.e.	In compliance with the Continuous Acquisition Lifecycle Support (CALS) program, JTR complies with specifications and standards approved within DOD for creation, use, and management of technical and other data in digital form.		N	No		

ORD						
ORD Paragraph Number	OPERATIONAL REQUIREMENTS DOCUMENT (ORD) For JOINT TACTICAL RADIO (JTR)	Requirement Category - KPP or Threshold	Identified as High Arch Driver in Step 1	Architecture Driver: High, Med, Low	SCAS Reference	SRD TEXT
5.e.	For Army and Air Force employment, logistics support should include sufficient quantities of Mobility Readiness Spares packages and Peacetime Operating Stocks for continued supportability. For the Navy and Marine Corps, spares is based on the On Board Repair Parts requirements, as calculated for each platform. If required, spares is pre-positioned.					
5.f.(2)	The JTR has adequate security safeguards and compartmentalization to ensure the confidentiality, integrity, and availability of the information passing through or residing on it. Security features of the JTR comply with the Multi-Level Information Systems Security Initiative (MISSI).		Yes	High	Security Supplement	Implementation of this capability has NSA participation and conforms to NSA guidelines as well as comply with MISSI.
5.f.(2)	Security features of the JTR comply with its follow-on.					
5.f.(2)	All C4I resources are certified for end-to-end interoperability by complying with the intent of CJCSI 6212.01A, 30 June 1995.					
5.f.(3)	This ORD has been assigned a joint potential designator (JPD) of "Joint."					
5.g.	JTR distribution and basing are consistent with existing force structures and deployment concepts. If JTR components are integrated into other systems, transportability requirements of the host system apply.					
5.h.(1)	The JTR acquisition adhere to the Joint Technical Architecture in identifying the standards and guidelines.					

ORD						
ORD Paragraph Number	OPERATIONAL REQUIREMENTS DOCUMENT (ORD) For JOINT TACTICAL RADIO (JTR)	Requirement Category - KPP or Threshold	Identified as High Arch Driver in Step 1	Architecture Driver: High, Med, Low	SCAS Reference	SRD TEXT
5.h.(2)	Electronic keying is applied to allow cryptographic processes implemented by JTR. All cryptographic systems must be interoperable with the EKMS, and the envisioned Joint Key Management System (JKMS).		N	Low	Sec 4.2.3.4	The SCAS does not define performance for specific waveforms or capabilities. Specifically, EKMS extends well beyond the scope for any specific JTR product. The SCAS does define a hardware INFOSEC Class that has a key fill type attribute that permits implementation of hardware to interface in an EKMS compliant manner when such compliance is defined.
5.h.(2)	Only National Security Agency (NSA) endorsed and approved security products, techniques, and protective services is used to secure classified communications.		N	No		
5.h.(3)	The JTR must be interoperable with National Airspace System architecture.					
5.h.(4)	For operation with North Atlantic Treaty Organization (NATO) member nations, the technical characteristics of the JTR conforms with the applicable requirements of the Standardization Agreements.		N	No		
5.h.(5)	All information technology for the JTR is DOD approved and where applicable, selected from those contained in the DISA approved "Profile of Standards."		N	Low	Sec 1.2.1; JTRS API Supplement	JTRS implementations are C4I systems that comply with JTA data exchange and network management protocols. These protocols provide for network link status. Applications waveforms that provide wireless networking capability are required to support network management requirements. These requirements are not part of the SCAS, but would be application waveform performance requirements.
5.i.	When required, JTR use National Imagery and Mapping Agency (NIMA) joint service mapping standards to ensure interoperability with other systems. Geographic mapping and gridding functions are based on Universal Transverse Mercator (UTM).					
5.i.	Latitude/longitude coordinates referred to by the World Geodetic System (WGS-84), are compatible with existing GPS receivers, and upgradable to future GPS receivers.					

ORD						
ORD Paragraph Number	**OPERATIONAL REQUIREMENTS DOCUMENT (ORD) For JOINT TACTICAL RADIO (JTR)**	**Requirement Category - KPP or Threshold**	**Identified as High Arch Driver in Step 1**	**Architecture Driver: High, Med, Low**	**SCAS Reference**	**SRD TEXT**
6	Refer to the domain annexes for the Services force structure requirements. Other governmental agencies' force structure requirements are described on a case by case basis.					
7.a.	The system is developed incrementally providing increased capabilities as it matures.		Yes	Med	Sec 3.1, 3.2, 4.5	Generally, the features of the SCAS that facilitate modularity and scalability also allows the system to be developed incrementally. These features include the use of both COTS hardware and software and the use of API's to define interface behavior of custom interfaces.
7.b.	The JTR supports the modes/capabilities depicted in Tables 1 and 2.					
7.d.	Delivery of follow-on production level articles, entailing additional objective values, are scheduled to begin in FY02 (Table 2).					
	The JTR network has the capacity to meet the information flow required by new capabilities and a latency near zero (objective).	Objective	No	High	Sec 3.2.2.1.2 and Figure 3-35.	Hardware limitations impose latency on information transfer within the system. Higher speed data rates are implemented by inserting new technology (as required) and by using alternate transfer mechanism as permitted by 3.2.2.1.2.
	The JTR provides a scaleable, embedded programmable cryptographic capability, which support black key [for the] Maritime/Fixed Station Domain (Objective).	Objective	N	Med	Sec 4.2.3.4	The SCAS does not define performance for specific waveforms or capabilities. The SCAS does define a hardware INFOSEC Class that has attributes sufficient for all required cryptographic functions.
	The JTR provides the ability to scan individual frequency bands (objective).	Objective	N	No		

Table 7-2: ORD Annex A

ANNEX A						
ORD Annex A Paragraph Number	**ORD Annex A Airborne Domain Requirements with Table 1&2 Waveform Requirements (changes from basic ORD)**	**Requirement Category - KPP or Threshold**	**Identified as High Arch Driver in Step 1**	**Architecture Driver: High, Med, Low**	**SCAS Reference**	**SRD Draft text**
4.a.(1)(a)	The airborne JTR meets required performance parameters when integrated into land and sea based fixed, rotary wing, and unmanned aircraft (threshold).	Threshold	N	No		
4.a.(1)(b) 1	The JTR provides interfaces for host platform visual displays (threshold).	Threshold	N	Low	Sec 2.2.1.7.6; 3.1.3	See rationale for ORD para 4.c.(4). The SCAS isolates user and platform interfaces through modularity of the software and hardware associated with those interfaces. Platform unique interfaces are accommodated within the User I/O hardware class and through extensions of the CF *Resource* class and the *Device* class such as I/O Device
4.a.(1)(b) 2	The JTR provides standard interfaces for host platform data input/output devices, including control and traffic platform busses (threshold).	Threshold	N	Low	Sec 2.2.1.7.6; 3.1.3, 4.2.3.5	The SCAS isolates user and platform interfaces through modularity of the software and hardware associated with those interfaces. Platform unique interfaces are accommodated within the User I/O hardware class and through extensions of the CF *Resource* class and the *Device* class such as I/O Device
4.a.(1)(b) 4	The airborne JTR provides for remote control and operation via a remote control unit or through the host platform bus interface (threshold).	Threshold	N	Low	3.2.2.1	Control of an Airborne JTRS is expected to be via a standard bus protocol or a legacy control interface. Both of these types of interfaces are supported in the SCAS by the API services defined in section 3.2.2.1
4.a.(1)(c)	The airborne JTR supports performance parameters while operating in the operational profile of each host airborne platform (threshold).	Threshold	N	No		
4.a.(1)(e)	In addition to GPS, the airborne JTR provides up to six channels (threshold).	Threshold	N	Low		The SCAS does not define performance required to meet waveform/capabilities. That depends on implementation hardware. The SCAS does not limit the number of waveforms which can be available for deployment.

ANNEX A

ORD Annex A Paragraph Number	ORD Annex A Airborne Domain Requirements with Table 1&2 Waveform Requirements (changes from basic ORD)	Requirement Category - KPP or Threshold	Identified as High Arch Driver in Step 1	Architecture Driver: High, Med, Low	SCAS Reference	SRD Draft text
4.a.(1)(f)	The airborne JTR provides the capability to choose from among at least 10 waveforms without loading additional software from an external source.(no KPP, Thres, or Obj assigned)	Threshold	N	Low	Sec 3.1.3.3.1; 3.1.3.3.2; 3.1.3.3.3	Waveform application software are contained in files stored on internal storage devices or external. The SCAS CF specifies the requirements for these files. The number of applications required to be resident within a JTRS implementation determines the capacity of the storage media for those files. The SCAS does not specify capacity.
4.a.(1)(f)	The airborne JTR provides the capability of replacing waveforms over-the-air.(no KPP, Thres, or Obj assigned)	Threshold	N	Low	3.1.3.3.2 Security Supplement	Waveform files may be transported over the air using established channels on JTRS. At the receiving JTRS node, the files received over-the air can be managed and stored using the file system of the Core Framework in conjunction with appropriate security considerations. Once stored, the local operator can select those waveform files for instantiation.
4.a.(1)(f)	The airborne JTR provides the capability to choose up to 30 waveforms using a bulk storage device.(no KPP, Thres, or Obj assigned)	Threshold	N	Low	3.1.3.3.2	Waveforms in JTRS are applications as defined in section 3.2. These applications are if the form of a set of file(s). These waveform application files are supported by a federated file system, which is part of the Core Framework defined in SCAS section 3.1.3.3.2. This federated file system allows the waveform files be located anywhere in the domain. This includes the mounting of an external Bulk storage device on to the domain.
4.c.(1)(a)	Integration of JTR into user platforms is accomplished with minimal demands for platform modifications. The JTR system provides radios and ancillaries that have the following characteristics:		N	Low	Sec 2.2.1.7.4 4.2.3.5	The SCAS does not define interfaces required to meet specific platform requirements. That depends on implementation hardware. The SCAS does define software resources and hardware classes enabling the implementation of platform-specific interfaces.
4.c.(1)(a) 1	The airborne JTR (without ancillaries) is no larger than a ¾ long air transportable rack (ATR) (threshold).	Threshold	N	No		
4.c.(1)(a) 1	The airborne JTR (without ancillaries) is no longer than a ½ air transportable rack (ATR) (objective).	Objective	N	No		
4.c.(1)(a) 2	The airborne JTR is no heavier than the radios it replaces (threshold).	Threshold	N	No		

ANNEX A

ORD Annex A Paragraph Number	ORD Annex A Airborne Domain Requirements with Table 1&2 Waveform Requirements (changes from basic ORD)	Requirement Category - KPP or Threshold	Identified as High Arch Driver in Step 1	Architecture Driver: High, Med, Low	SCAS Reference	SRD Draft text
4.c.(1)(a) 2	The airborne JTR is at least 75% lighter than the radios it replaces (objective).	Objective	N	No		
4.c.(1)(a) 3	The airborne JTR operates off existing aircraft power systems for each platform (threshold).	Threshold	N	Low	Sec 4.2.2	The SCAS isolates Power Supplies into a separate class, so that platform unique requirements can be separated from internal system power needs. This enable an airborne system acquisition authority to specify platform specific power and the JTRS system supplier to only have to change a single hardware class to accommodate the unique power of the platform.
4.c.(1)(a) 4	The airborne JTR draws no more power than the radios it replaces (threshold).	Threshold	N	No		
4.c.(1)(a) 4	The airborne JTR draws at least 75% less power than the radios it replaces (objective).	Objective	N	No		
4.c.(1)(a) 5	The airborne JTR provides interfaces to on-board automated frequency management systems (threshold).	Threshold	N	Low	Sec 2.2.1.7.6; 3.1.3, 4.2.3.5	The SCAS isolates user and platform interfaces through modularity of the software and hardware associated with those interfaces. Platform unique interfaces are accommodated within the User I/O hardware class and through extensions of the CF *Resource* class and the *Device* class such as I/O Device
4.c.(2)(a)	The JTR is capable of being operated and maintained in a nuclear, biological, and chemical (NBC) environment by persons in full Mission Oriented Protective Posture IV (MOPP IV) protection gear (threshold).	Threshold	N	No		
4.c.(2)(b)	The JTR man-machine interface is compatible with Night Vision Imaging System (NVIS) standards and is capable of being operated by persons wearing night vision goggles (NVG) (threshold).	Threshold	N	No		

Table 7-3: ORD Annex B

ANNEX B						
ORD Annex B Paragraph Number	**ORD Annex B_Maritime/Fixed Domain_Requirements with Table 1&2 Waveform Requirements**	**Requirement Category - KPP or Threshold**	**Identified as High Arch Driver in Step 1**	**Architecture Driver: High, Med, Low**	**SCAS Reference**	**SRD TEXT**
1.b.	The maritime JTR is part of a communications system that provides modular communicating and networking capabilities.		N	Low	3.2.1, 3.2.2.1	3.2.1 The SCAS supports the hosting of various networking protocols as applications. Seamless transfer of data is supported through the use of the Service API's. Service API's include Network, Security and I/O services for the domain.
1.c.	Operational Concept (Supplemental). The maritime JTR is Automated Digital Network System (ADNS) interoperable.		N	Low	Sec 2.2.1.7.4 4.2.3.5	The SCAS does not define interfaces required to meet specific platform requirements. That depends on implementation hardware. The SCAS does define resources and hardware classes enabling the implementation of platform-specific interfaces.
1.c.	The maritime JTR is Advanced RF distribution systems interoperable.		N	Low	Sec 4.2.3.1	The hardware classes of the SCAS isolates RF external interfaces to the RF class. The Maritime domain acquisition authority would specify the interface requirements for the Advanced RF distribution systems and the system supplier would isolate those requirements to their implementation of the RF class.
1.c.	The maritime JTR is Submarine Antenna Distribution System (SADS) interoperable (i.e. JMCOMS interoperable).		N	Low	Sec 4.2.3.1	The hardware classes of the SCAS isolates RF external interfaces to the RF class. The Maritime domain acquisition authority would specify the interface requirements for the Advanced RF distribution systems and the system supplier would isolate those requirements to their implementation of the RF class.
4.a.	The JTR meets the following supplemental maritime specific performance parameters:					
4.a.(1)(a)	In addition to GPS, the Maritime and Fixed JTR provides a minimum of 4 scaleable channels (threshold).	Threshold	N	Low	4.5.1, Appendix B.3	Number and availability of channels is dependent on domain (Handheld, Maritime, etc). Simultaneous operation of multiple channels is a function of the performance of the domain hardware. A multi-process OS is defined by the SCAS and is detailed in appendix B.3
4.a.(1)(a)	In addition to GPS, the Maritime and Fixed JTR provides a minimum of 4 scaleable channels with a growth capability to 10 scaleable channels (objective).	Objective	N	Med		The SCAS does not define performance required to meet waveform/capabilities. That depends on implementation hardware.

ANNEX B

ORD Annex B Paragraph Number	ORD Annex B_Maritime/Fixed Domain_Requirements with Table 1&2 Waveform Requirements	Requirement Category - KPP or Threshold	Identified as High Arch Driver in Step 1	Architecture Driver: High, Med, Low	SCAS Reference	SRD TEXT
4.a.(1)(b)	The JTR is capable of operating in sea state 5 and surviving sea state 8 on all classes of ships (threshold).	Threshold	N	No		
4.a.(1)(b)	The JTR is capable of operating at sea states above 5 with minimal degraded performance (objective).	Objective	N	No		
4.a.(1)(d)	The Maritime and Fixed station configuration of JTR provides the capability for radios to be operated, controlled, and monitored from remote locations (threshold).	Threshold	N	Low	Sec 2.2.1.7.4 4.2.3.5	The SCAS does not define interfaces required to meet specific platform requirements. That depends on implementation hardware. The SCAS does define software resources and hardware classes enabling the implementation of platform-specific interfaces.
4.a.(1)(e)	The JTR is compatible with commercial, ground mobile, and shipboard power distributed systems (threshold).	Threshold	N	No		
4.a.(1)(g)	The JTR weight does not exceed a two person lift (threshold).	Threshold	N	No		
4.a.(1)(h)	The JTR has a minimum of 10 presets per channel (threshold).	Threshold	N	No		
4.a.(1)(h)	The JTR has a minimum of 20 presets per channel (objective).	Objective	N	No		
4.a.(1)(i)	The JTR provides a standard interface with legacy shipboard and fixed station communication systems (threshold).	Threshold	N	Low	Sec 2.2.1.7.6; 3.1.3	See rationale for ORD para 4.c.(4). The SCAS isolates user and platform interfaces through modularity of the software and hardware associated with those interfaces. Platform unique interfaces are accommodated within the User I/O hardware class and through extensions of the CF *Resource* class and the *Device* class such as I/O Device
4.a.(1)(j)	The Maritime/Fixed Station JTR provides the capability of replacing waveforms over-the-air (threshold).	Threshold	N	Low	3.1.3.3.2 Security Supplement	Waveform files may be transported over the air using established channels on JTRS. At the receiving JTRS node, the files received over-the air can be managed and stored using the file system of the Core Framework in conjunction with appropriate security considerations. Once stored, the local operator can select those waveform files for instantiation.

ANNEX B						
ORD Annex B Paragraph Number	**ORD Annex B_Maritime/Fixed Domain_Requirements with Table 1&2 Waveform Requirements**	**Requirement Category - KPP or Threshold**	**Identified as High Arch Driver in Step 1**	**Architecture Driver: High, Med, Low**	**SCAS Reference**	**SRD TEXT**
4.a.(1)(j)	The Maritime/Fixed Station JTR provides the capability to choose up to 30 waveforms using a bulk storage device (threshold).	Threshold	N	Low	3.1.3.3.2	Waveforms in JTRS are applications as defined in section 3.2. These applications are if the form of a set of file(s). These waveform application files are supported by a federated file system, which is part of the Core Framework defined in SCAS section 3.1.3.3.2. This federated file system allows the waveform files be located anywhere in the domain. This includes the mounting of an external Bulk storage device on to the domain.
4.a.(1)(j)	The Maritime/Fixed Station JTR provides the capability to choose from among at least 12 waveforms without loading additional software from an external source (threshold).	Threshold	N	No		
4.c.(1)	The JTR is capable of being operated in low light shipboard conditions (threshold).	Threshold	N	No		
4.c.(2)	Temperature constraints conforms with best commercial practices (threshold).	Threshold	N	No		
5.b.	JTR utilizes the Consolidated Automated Support System (CASS).	Threshold	N	No		
5.c.	All final manpower, personnel, and training (MPT) requirements is documented in Service training plans.	Threshold	N	No		

Table 7-4: ORD Annex C

ANNEX C						
ORD Annex C Paragraph Number	**ORD Annex C_Ground Forces Domain_Requirements with Table 1&2 Waveform Requirements**	**Requirement Category - KPP or Threshold**	**Identified as High Arch Driver in Step 1**	**Architecture Driver: High, Med, Low**	**SCAS Reference**	**SRD TEXT**
4	All ground forces requirements for JTR, listed below, is based on validated rules for Operational Facilities (OPFACs) and validated Information Exchange Requirements (IERs), between OPFACs contained in the US Army Training and Doctrine Command (TRADOC) Command, Control, Communications, and Computers Requirements Definition Program (C4RDP), or Joint equivalent.		N	No		
4.a.(1)(b)	The vehicular and dismounted warfighter configurations of the JTR provides the capability for radios to be operated and controlled from remote locations up to 2km away (threshold).	Threshold	N	Low	Sec 2.2.1.7.4 4.2.3.5	The SCAS does not define interfaces required to meet specific platform requirements. That depends on implementation hardware. The SCAS does define software resources and hardware classes enabling the implementation of platform-specific interfaces.
4.a.(1)(b)	The vehicular and dismounted warfighter configurations of the JTR provides the capability for radios to be operated and controlled from remote locations up to 4km away (objective).	Objective	N	Low	3.2.2.1	Control of an vehicular or dismounted JTR is via a standard bus protocol or a legacy control interface. Both of these types of interfaces are supported in the SCAS by the API services defined in section 3.2.2.1.
4.a.(1)(c)	A JTR operates at full performance levels and not degrade mission effectiveness of host systems/platforms engaged in their operational environments, including movement and weapons firing (threshold) (KPP).	KPP	N	No		
4.a.(1)(d)	The JTR survives High Altitude Electromagnetic Pulse (HEMP) to the degree specified in MIL-STD-2169B but not be required to work through the event (threshold). Recycling of power to restore operation is acceptable.	Threshold	N	No		
4.a.(1)(e)	The JTR survives chemical, and biological attacks as well as decontamination procedures using existing solvents	Threshold	N	No		

ANNEX C

ORD Annex C Paragraph Number	ORD Annex C_Ground Forces Domain_Requirements with Table 1&2 Waveform Requirements	Requirement Category - KPP or Threshold	Identified as High Arch Driver in Step 1	Architecture Driver: High, Med, Low	SCAS Reference	SRD TEXT
	(threshold).					
4.a.(1)(f)	The JTR provides a display of current own position location information at each radio (threshold).	Threshold	N	No		
4.a.(1)(g)	The JTR system provides operator selectable display modes that express its position in either the GPS latitude-longitude or the Military Grid Reference System (MGRS), that includes a 3-character grid zone, a 2-character 100km square, and an 8-digit map coordinate (threshold).	Threshold	N	No		
4.a.(1)(g)	The JTR system provides operator selectable display modes that express its position in either the GPS latitude-longitude or the Military Grid Reference System (MGRS), that includes a 3-character grid zone, a 2-character 100km square, and an 10-digit map coordinate (objective).	Objective				
4.a.(1)(h)	The JTR is operable and maintainable in temperatures from -40°C to +55°C (threshold).	Threshold	N	No		
4.a.(1)(i)	The JTRs provides the means to physically interconnect to selected external legacy radios to access the JTR network (threshold).	Threshold	N	Low	Sec 2.2.1.7.4 4.2.3.5	The SCAS does not define interfaces required to meet specific platform requirements. That depends on implementation hardware. The SCAS does define resources and hardware classes enabling the implementation of platform-specific interfaces.
4.a.(1)(j)	The ground forces JTR, in the Handheld configuration, provides the capability to choose from among at least 6 waveforms without loading additional software from an external source (threshold).	Threshold	N	High	Sec 3.1.3.3.1; 3.1.3.3.2; 3.1.3.3.3	Waveform application software are contained in files stored on internal storage devices or external. The SCAS CF specifies the requirements for these files. The number of applications required to be resident within a JTRS implementation determines the capacity of the storage media for those files. The SCAS does not specify capacity. This is considered a high architecture driver for this domain because of very severe size, weight and power constraints on handheld implementations. Technology insertion of new denser and lower power electronics reduces the impact on implementing JTRS in the handheld domain.

ANNEX C

ORD Annex C Paragraph Number	ORD Annex C_Ground Forces Domain_Requirements with Table 1&2 Waveform Requirements	Requirement Category - KPP or Threshold	Identified as High Arch Driver in Step 1	Architecture Driver: High, Med, Low	SCAS Reference	SRD TEXT
4.a.(1)(j)	The ground forces JTR, in the Handheld configuration, provides the capability of replacing waveforms over-the-air (threshold).	Threshold	N	High	3.1.3.3.2 Security Supplement	Waveform files may be transported over the air using established channels on JTRS. At the receiving JTRS node, the files received over-the air can be managed and stored using the file system of the Core Framework in conjunction with appropriate security considerations. Once stored, the local operator can select those waveform files for instantiation.
4.a.(1)(j)	The ground forces JTR, in the Handheld configuration, provides the capability to choose up to 30 waveforms using a bulk storage device (threshold).	Threshold	N	High	3.1.3.3.2	Waveforms in JTRS are applications as defined in section 3.2. These applications are if the form of a set of file(s). These waveform application files are supported by a federated file system, which is part of the Core Framework defined in SCAS section 3.1.3.3.2. This federated file system allows the waveform files be located anywhere in the domain. This includes the mounting of an external Bulk storage device on to the domain.
4.a.(1)(j)	The ground forces JTR, in the Vehicular configuration, provides the capability to choose from among 10 waveforms without loading additional software from an external source (threshold).	Threshold	N	Low	Sec 3.1.3.3.1; 3.1.3.3.2; 3.1.3.3.3	Waveform application software are contained in files stored on internal storage devices or external. The SCAS CF specifies the requirements for these files. The number of applications required to be resident within a JTRS implementation determines the capacity of the storage media for those files. The SCAS does not specify capacity.
4.a.(1)(j)	The ground forces JTR, in the Vehicular configuration, provides the capability of replacing waveforms over-the-air (threshold).	Threshold	N	Low	3.1.3.3.2 Security Supplement	Waveform files may be transported over the air using established channels on JTRS. At the receiving JTRS node, the files received over-the air can be managed and stored using the file system of the Core Framework in conjunction with appropriate security considerations. Once stored, the local operator can select those waveform files for instantiation.
4.a.(1)(j)	The ground forces JTR, in the Vehicular configuration, provides the capability to choose up to 30 waveforms using a bulk storage device (threshold).	Threshold	N	Low	3.1.3.3.2	Waveforms in JTRS are applications as defined in section 3.2. These applications are if the form of a set of file(s). These waveform application files are supported by a federated file system, which is part of the Core Framework defined in SCAS section 3.1.3.3.2. This federated file system allows the waveform files be located anywhere in the domain. This includes the mounting of an external Bulk storage device on to the domain.

ANNEX C

ORD Annex C Paragraph Number	ORD Annex C_Ground Forces Domain_Requirements with Table 1&2 Waveform Requirements	Requirement Category - KPP or Threshold	Identified as High Arch Driver in Step 1	Architecture Driver: High, Med, Low	SCAS Reference	SRD TEXT
4.a.(1)(j)	The ground forces JTR, in the Dismounted Warfighter configuration, provides the capability to choose from among 10 waveforms without loading additional software from an external source (threshold).	Threshold	N	Med	Sec 3.1.3.3.1; 3.1.3.3.2; 3.1.3.3.3	Waveform application software are contained in files stored on internal storage devices or external. The SCAS CF specifies the requirements for these files. The number of applications required to be resident within a JTRS implementation determines the capacity of the storage media for those files. The SCAS does not specify capacity.
4.a.(1)(j)	The ground forces JTR, in the Dismounted Warfighter configuration, provides the capability of replacing waveforms over-the-air (threshold).	Threshold	N	Med	3.1.3.3.2 Security Supplement	Waveform files may be transported over the air using established channels on JTRS. At the receiving JTRS node, the files received over-the air can be managed and stored using the file system of the Core Framework in conjunction with appropriate security considerations. Once stored, the local operator can select those waveform files for instantiation.
4.a.(1)(j)	The ground forces JTR, in the Dismounted Warfighter configuration, provides the capability to choose up to 30 waveforms using a bulk storage device (threshold).	Threshold	N	Med	Sec 3.1.3.3.1; 3.1.3.3.2; 3.1.3.3.3	Waveform application software are contained in files stored on internal storage devices or external. The SCAS CF specifies the requirements for these files. The number of applications required to be resident within a JTRS implementation determines the capacity of the storage media for those files. The SCAS does not specify capacity.
4.b.	The JTR is transportable worldwide (air, rail, sea, and air droppable) (threshold).	Threshold	N	No		
4.c.(1)	Integration of JTR radios and ancillaries (e.g., installation kits power amplifiers, and antennae) into user platforms is accomplished with minimal demands for platform modifications.	Threshold	N	Low	Sec 2.2.1.7.4 4.2.3.5	The SCAS does not define interfaces required to meet specific platform requirements. That depends on implementation hardware. The SCAS does define resources and hardware classes enabling the implementation of platform-specific interfaces. The SCAS does not limit the number of waveforms which can be available for deployment.
4.c.(1)(a)	The vehicular JTR is smaller than the radio and ancillary equipment that it replaces (threshold).	Threshold	N	No		
4.c.(1)(a)	The vehicular JTR is75% smaller than the radio and ancillary equipment that it replaces (objective).	Objective	N	No		

ANNEX C

ORD Annex C Paragraph Number	ORD Annex C_Ground Forces Domain_Requirements with Table 1&2 Waveform Requirements	Requirement Category - KPP or Threshold	Identified as High Arch Driver in Step 1	Architecture Driver: High, Med, Low	SCAS Reference	SRD TEXT
4.c.(1)(b)	The handheld JTR, including the ancillary equipment, is no larger than the size of comparable existing handheld land mobile radios (threshold).	Threshold	N	High	Sec 3.1	This is considered a high architecture driver for this domain because of very severe size, weight and power constraints on handheld implementations. Requiring handheld systems to meet all interface requirements of the operating environment places additional processing requirements on implementations limited in size, weight and power. Technology insertion of new denser and lower power electronics reduces the impact on implementing JTRS in the handheld domain. Step 2B prototyping efforts further define capabilities and limitations possible for the handheld domain of JTRS.
4.c.(1)(b)	The handheld JTR, including the ancillary equipment, is no larger than the size of comparable existing handheld land mobile radios and be capable of being integrated into Land Warrior electronic component housing (objective).	Objective	N	High	Sec 3.1	This is considered a high architecture driver for this domain because of very severe size, weight and power constraints on handheld implementations. Requiring handheld systems to meet all interface requirements of the operating environment places additional processing requirements on implementations limited in size, weight and power. Technology insertion of new denser and lower power electronics reduce the impact on implementing JTRS in the handheld domain. Step 2B prototyping efforts further define capabilities and limitations possible for the handheld domain of JTRS.
4.c.(1)(c)	The dismounted warfighter JTR, including the ancillary equipment, doesnot exceed 400 cubic inches (threshold).	Threshold	N	No		
4.c.(1)(c)	The dismounted warfighter JTR, including the ancillary equipment, does not exceed 200 cubic inches (objective).	Objective	N	No		
4.c.(1)(d)	The vehicular JTR weighs less than the radios and ancillary equipment that it replaces (threshold).	Threshold	N	No		
4.c.(1)(d)	The vehicular JTR weighs 75% less than the radios and ancillary equipment that it replaces (objective).	Objective	N	No		
4.c.(1)(e)	The handheld JTR, including battery and antenna, is no more than 3 pounds (threshold).	Threshold	N	No		
4.c.(1)(e)	The handheld JTR, including battery and antenna, is no more than one pound (objective).	Objective	N	No		

ANNEX C

ORD Annex C Paragraph Number	ORD Annex C_Ground Forces Domain_Requirements with Table 1&2 Waveform Requirements	Requirement Category - KPP or Threshold	Identified as High Arch Driver in Step 1	Architecture Driver: High, Med, Low	SCAS Reference	SRD TEXT
4.c.(1)(f)	The dismounted warfighter JTR, including ancillary equipment, does not exceed 13 pounds (threshold).	Threshold	N	No		
4.c.(1)(f)	The dismounted warfighter JTR, including ancillary equipment, does not exceed 6 pounds (objective).	Objective	N	No		
4.c.(2)(a)	JTRs draws no more primary power than the radios and ancillary equipment replaced (threshold).	Threshold	N	No		
4.c.(2)(a)	JTRs draws at least 75% less power than the radios and ancillary equipment replaced (objective).	Objective	N	No		
4.c.(2)(b)	The JTR is capable of being operated with primary power derived from batteries and DC power systems (threshold).	Threshold	N	No		
4.c.(2)(b)	The JTR is capable of being operated with primary power derived from batteries and DC power systems and from new power systems (objective).	Objective				
4.c.(3)(a)	In addition to GPS, the vehicular JTR provides up to five channels (threshold).	Threshold	N	No		
4.c.(3)(a)	In addition to GPS, the vehicular JTR provides up to eight channels (objective).	Objective	N	No		
4.c.(3)(b)	In addition to GPS, the dismounted warfighter JTR provides two channels (threshold).	Threshold	N	No		
4.c.(3)(b)	In addition to GPS, the dismounted warfighter JTR provides up to four channels (objective).	Objective	N	No		
4.c.(3)(c)	In addition to GPS, the handheld JTR provides one channel (threshold).	Threshold	N	No		
4.c.(3)(c)	In addition to GPS, the handheld JTR provides two channels (objective).	Objective	N	No		

ANNEX C

ORD Annex C Paragraph Number	ORD Annex C_Ground Forces Domain_Requirements with Table 1&2 Waveform Requirements	Requirement Category - KPP or Threshold	Identified as High Arch Driver in Step 1	Architecture Driver: High, Med, Low	SCAS Reference	SRD TEXT
4.c.(4)	Each JTR provides access to auxiliary data and voice/video/data access on each channel (threshold).	Threshold	N	Low	Sec 4.2.3; 1.2.1; JTRS API Supplement	The H/W class, I/O, isolates user interfaces to that entity. The acquisition authority can specify the input and output requirements for each channel of the system and the supplier implements the requirement on this hardware I/O class. The JTA imposes standard interfaces for these data exchange types. No specific requirements on the external interfaces of the waveform application are included in the SCAS. However, the interface standards imposed by the JTA apply to JTRS as a C4I system.
4.c.(5)(a)	Provides for safe, efficient and effective operation and maintenance by normal and typically trained personnel while wearing any combination of night vision devices, MOPP IV gear, and cold weather protective gear (threshold).	Threshold	N	No		
4.c.(5)(b)	Adheres to the guidance of applicable Military Standards intended to preclude or minimize exposure to health hazards and threats to soldier survivability (threshold).	Threshold	N	No		

Table 7-5: Step 1 Program Objectives

STEP 1 PROGRAM OBJECTIVES						
RFP Solicitation Ref	**Program Objective (Step 1 Evaluation Criteria**	**Requirement Category - KPP or Threshold**	**Identified as High Arch Driver in Step 1**	**Architecture Driver: High, Med, Low**	**SCAS Reference**	**SRD TEXT**
4.1	Maximizes independence of software from specific hardware solutions	Prog Obj	na	High	Sec 3.1.1; 3.1.2; 3.2.1.1; 3.2.1.2	Making software independent from hardware requires the software to be isolated from the processors in the system through a common interface that can be implemented on many processors. Within the SCAS, specifying POSIX and CORBA middleware for the OE and for applications performs that isolation. Both these standards have multiple commercial products that can be implemented on many different processors.
4.1	Encourages industry acceptance as a commercial standard	Prog Obj	na	High	Foreword, 2.2.1, 3.1.1, 3.2.1, 3.2.2., 4.5.2.2, 4.5.3	The SCAS either requires or strongly encourages the use commercial standards for software and hardware design.
4.1	Is scaleable from low-capability hand-held equipment to high-capability fixed-station equipment	Prog Obj	na	High	3.1.1, 3.2, 4.5	In software designs using object techniques, scalability through simple method/operation additions and deletions is common practice. Common hardware modules can populate different chassis. The number of modules in a chassis is transparent to domain control. Only the number of channels and their resources are visible. Scalability also takes the form of changing an implementation from an FPGA to an ASIC.
4.1	Incorporates information security (INFOSEC)	Prog Obj	na	High	Sec 2.2.1.7.5 4.2.3.4; Security Supplement	The SCAS defines software resources and hardware classes that permit the incorporation of INFOSEC capabilities. Architectural definition for multilevel security requirements of a multiple, simultaneous channel system have not been determined yet and therefore this is a high architectural driver.
4.1	Addresses the JTRS requirements contained in the JTRS Operational Requirements Document (ORD).	Prog Obj	na	High	Entire SCAS v 2.2	The ORD to SCAS traceability provided in the previous 4 spreadsheets address this requirement. It is considered a high driver because there are individual ORD requirements that are high drivers.

STEP 1 PROGRAM OBJECTIVES

RFP Solicitation Ref	Program Objective (Step 1 Evaluation Criteria	Requirement Category - KPP or Threshold	Identified as High Arch Driver in Step 1	Architecture Driver: High, Med, Low	SCAS Reference	SRD TEXT
4.1	Yields an interoperable family of radios	Prog Obj	na	Low	Forward, 1, 1.2, 2.2.2.1, 3.2.2.1f, API Supplement	A major focus of the entire JTRS software architecture is to facilitate interoperability. The API features of the architecture have been established to support interoperability.
4.1	Yields a programmable and re-programmable family of radios	Prog Obj	na	Low	3 (all)	A major focus of the entire JTRS software architecture is to facilitate re-programmability. The SCAS defines the Operating Environment (core framework, CORBA, and Operating System) and specifies the services and interfaces that applications use from that environment. The Operating Environment imposes design constraints on waveform and other applications to provide increased portability of those applications from one SCA-compliant platform to another. These design constraints include specified interfaces between the Core Framework and application software, and restrictions on waveform usage of the OS. This approach also provides a building block structure for defining APIs between application software components. This building block structure facilitates component-level reuse and allows significant flexibility for developers to define waveform-specific APIs. This approach also allows tailoring to meet the broad needs across the JTRS domains.
4.1	Is extendible to new waveforms and/or hardware components	Prog Obj	na	Low	Sec 3.1.3.1.5; 3.1.3.2.4; 4.2.2; 4.2.3	The SCAS defines software resources and hardware classes that permit the incorporation of future capabilities.
4.1	Provides rapid technology insertion for new hardware and software technologies that become available over time	Prog Obj	na	Med	SCAS v1.0 Forward; Sec 4.1	The partitioning of hardware elements into classes isolates functional modules and permits insertion of technology in one module without requiring changes to another. An example of this in the current prototyping is the availability of COTS processing boards with greater MIPS now than when the program started a year ago. That board can be directly inserted into the system. Technology insertion for software may require changes to the SCAS and some of its definitions to take full advantage of new software capabilities. That change process is in place through and SCAS CCB.

STEP 1 PROGRAM OBJECTIVES						
RFP Solicitation Ref	**Program Objective (Step 1 Evaluation Criteria**	**Requirement Category - KPP or Threshold**	**Identified as High Arch Driver in Step 1**	**Architecture Driver: High, Med, Low**	**SCAS Reference**	**SRD TEXT**
4.1	Provides an affordable family of radios	Prog Obj	na	Med	Forward, 1.1, 4.3	The JTRS architecture fosters affordability through use of commercial software and hardware standards as well as software and hardware reuse to the largest extent possible.